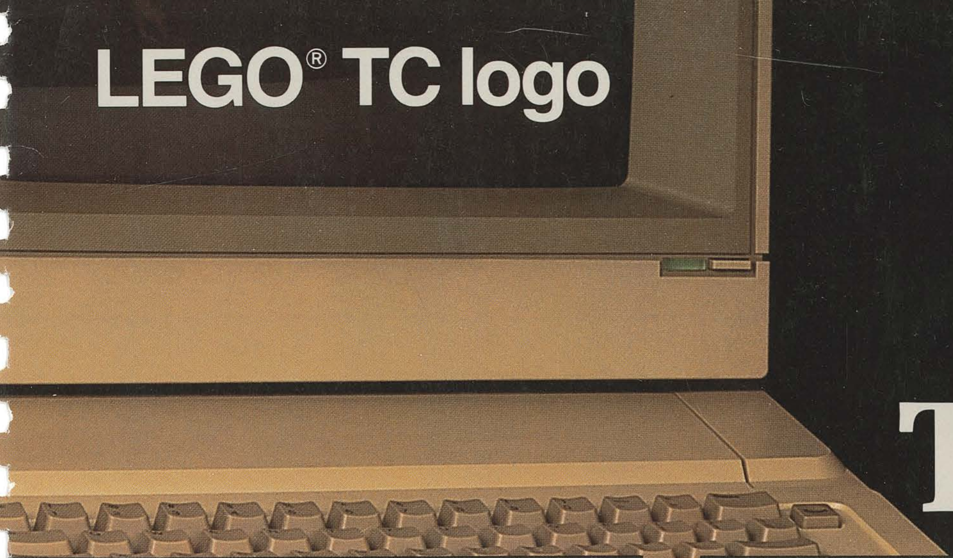
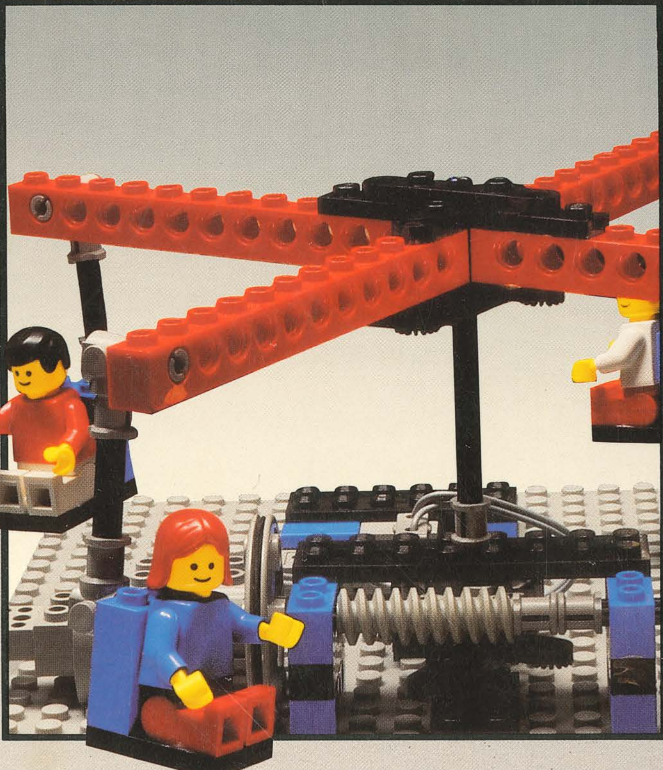
LEGO® TC logo

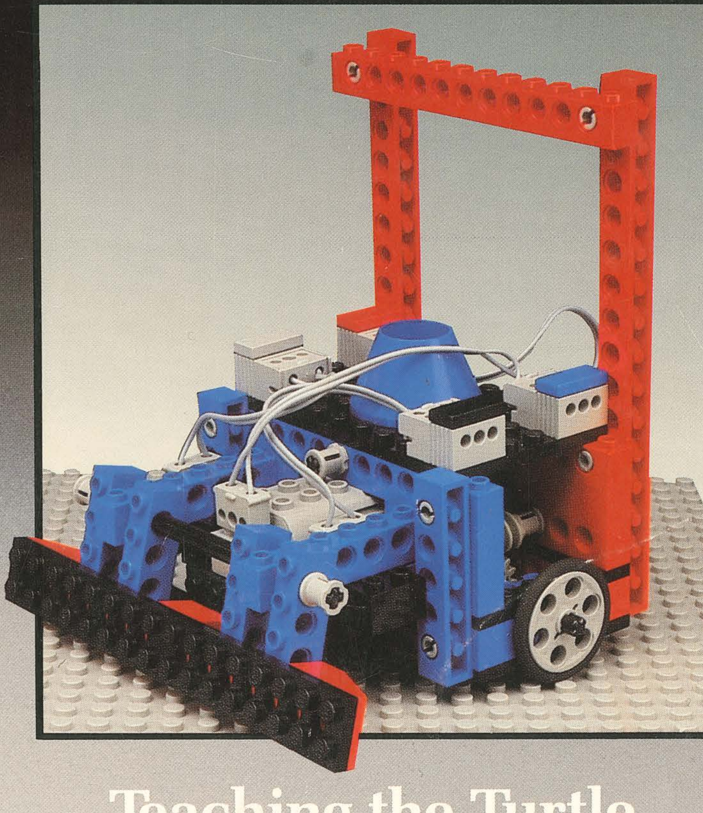


Teacher's Guide

Getting Started



Making Machines



Teaching the Turtle

LEGO dacta

LEGO® TC logo

Teacher's Guide

No part of the documents contained herein may be reproduced, stored in retrieval systems or transmitted in any form or by any means, photocopying, electronic, mechanical, recording or otherwise, without the prior approval in writing from the LEGO Group.

ISBN 0-914831-71-2

2nd revision, January 1989

LEGO® and the LEGO logo® are registered trademarks of INTERLEGO AG. © 1987, 1989 LEGO Group.

Apple and the Apple logo are registered trademarks of Apple Computer, Inc.

IBM is a registered trademark of International Business Machines Corporation.

MS-DOS is a registered trademark of Microsoft Corporation.

Development Team

Project Concept
Seymour Papert, Massachusetts Institute of Technology

Project Directors
Stephen Ocko, Microworlds Learning Inc.
Mitchel Resnick, Microworlds Learning Inc.

Research and Evaluation
Edith Ackermann, Massachusetts Institute of Technology
Bob Bickford, Independent consultant
Cathy Carrington, Strong School, Durham, Connecticut
Marilyn Schaffer, University of Hartford

Teacher Consultants
Adam Artis, William M. Trotter School, Boston, Massachusetts
Vincent Baldari, P. S. 380, Brooklyn, New York
Felice Cherry-Wingate, Frank D. Whalen Junior High School, Bronx, New York
Patricia Driscoll, Batchelder School, Hartford, Connecticut
Patricia Holohan, Annie Fisher School, Hartford, Connecticut
Linda Moriarty, James Hennigan School, Jamaica Plain, Massachusetts
Joanne Ronkin, James Hennigan School, Jamaica Plain, Massachusetts
Rosemary Volpe, Frank D. Whalen Junior High School, Bronx, New York
Alma Wright, William M. Trotter School, Boston, Massachusetts

Project Coordinators
Apple version: Elizabeth Gilbert and Catherine Helgoe, LEGO Dacta
IBM version: Catherine Helgoe, LEGO Dacta

Table of Contents

Introduction by Seymour Papert 5

Chapter 1
What is LEGO® TC logo? 11

Chapter 2
Setting Up 12

Chapter 3
Focusing on Projects 16

Chapter 4
Routes Through the Material 18

Chapter 5
Getting Familiar with LEGO® TC logo 23

Chapter 6
Teacher's Notes for *Getting Started* 29

Chapter 7
Teacher's Notes for *Making Machines* 85

Chapter 8
Teacher's Notes for *Teaching the Turtle* 113

Appendix 1
Worksheets 143

Bibliography 151

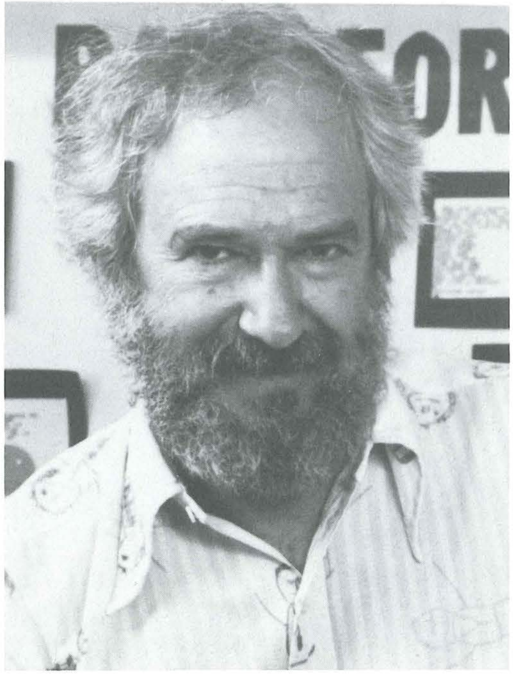
Introduction

I have never seen such enthusiasm from visiting educators as in the "LEGO® Room" at the Hennigan School, where we first tested LEGO® TC logo in the classroom. My plan, when I sat down to write this preface, was to find a few pithy words to explain what lies behind this enthusiasm. I soon realized that the plan had a bug: not just **one** wonderful thing was going on in the LEGO® Room, but many. My right brain buzzed with phrases like multi-faceted learning experience . . . cuts across differences in educational approach . . . allows expression of diverse learning styles. At the same time, my left brain flashed like a kaleidoscope with images of a hundred wonderful things children had done with LEGO® TC logo. All these thoughts and images are good in themselves, but I wanted to say something much more focused and specific; something that would actually help you use LEGO® TC logo. The idea of "capturing the essence of the LEGO® TC logo experience" came to seem, as I stared at my blank computer screen, like a hopelessly complex task.

Then, I had a breakthrough. I thought of those children and of the very complex things some of them had built out of LEGO® bricks. They were not daunted by complexity! Indeed, one of the wonderful things to be learned in working with LEGO® TC logo is the skill of project management; that is, the ability to make a complex thing happen. I believe that it should be possible to apply such skills to almost anything, ranging from mini-projects like building with LEGO® bricks, to super-projects like building a career or a life-style. Then I asked myself, why am I in this writing block? Why don't I use these "project skills" to write my preface? Why don't skills learned from building with LEGO® bricks transfer to building a preface with words?

They do. But, you as teachers surely know that transfer is not something that just happens. It is something that has to be made. It is an activity. It needs work. I broke out of my writing block when I actively sought similarities between the present problem (how to write a preface) and an old experience (working with LEGO® bricks). To refresh my memory, I wandered downstairs looking for LEGO® parts. Soon I was sitting at a table building tricycles!

I have a reaction to LEGO® parts that many children have: I have to take some in my hands. As soon as they are in my hands, they talk to me. They ask to be made into something. So, I begin with something exceedingly simple, and then intellectual juices flow and I find myself making it more complex, or more interesting, or maybe just changing it in some way so that I'll understand it better. So, I embroidered my tricycles by giving them motors and a way to steer. Then I began to focus on a LEGO® TC logo idea that had intrigued me months before: using the same motor to both drive and steer a LEGO® vehicle. When I had seen this, I had been awed. Now, as I did it myself, in my own way, I understood the idea and improved and extended it.



Please don't be upset if you do not care about power steering in LEGO® vehicles. As you work with LEGO® TC logo and watch children working with it, you will find out that this medium is wonderfully adaptive to many different tastes and styles. The point of my story has to do not with a particular project, but with a process. The LEGO® TC logo material incites you to give something you know very well (in my case, a tricycle) a new form (the LEGO® model). Doing so opens your mind to new aspects which you have not fully understood. By adding the dimension of computer control — the Logo aspect of LEGO® TC logo — you end up with a learning-rich, innovative project. LEGO® TC logo projects are rooted in your own experience and in things so familiar that they seem too simple to be worth thinking about. To my taste such projects are the best formula for learning.

* * * * * * * *

I was in a writing block because I wanted to do something complex. I wanted to capture the essence of LEGO® TC logo or read the minds of eminent educators who like LEGO® TC logo. I broke the block by doing something simple, something familiar and down to earth. My advice to teachers is to avoid being intimidated by what you think your students should be doing with LEGO® TC logo. Instead, help the children follow their own instinct for learning. They will do so in very personal ways.

I'd like to tell another story to show what I mean by "personal." The story illustrates how teachers must balance between too much structure (thus stifling spontaneous learning) and imposing too little structure (thus fostering chaos).

This story is about four third-grade girls at the Hennigan School. Everything about their bearing and self-image was "feminine" in a pre-women's movement sense. Their sense of what is appropriate for girls had been acquired from families and ethnic neighborhoods and had survived years of gender-neutral schooling. These girls knew some Logo and had used it to draw colored symmetry patterns on the screen. Their introductory activity in LEGO® TC logo was building a car (no motors, no computers) which they ran down an inclined track. Then, they were told they could work on any LEGO® project they would like.

Working in the LEGO® room, the girls saw many samples of models built by previous classes. Most models were highly mechanized with prominent signs of gears, motors and computer connections. How would you have expected them to respond?

The girls were obedient and would have built a motorized truck if told to do so. But they were given another option: freedom to choose. They began by choosing to do with LEGO® bricks what felt familiar and comfortable: rather than build motorized models, the girls built an ornately decorated house. Thus, the girls constructed a classical divide between "gentle artists" and "macho technologists." Observers watched impatiently: would the girls cross the line and how?

They did, and in ways that are richly suggestive for thinking about issues of science, gender and cultures, as well as for thinking about the role of teachers in LEGO® TC logo. One day the girls wrote the simplest possible Logo program: turn the light on, turn the light off, turn the light on, turn the light off. One day deep inside their house, a tiny light blinked.

This little blinking light may be an event of importance in the development of these girls. As I see it, they had found their way to take hold of the technical and scientific side of our world. They had timidly yearned all their lives to make this side of the world their own, but had been held back by deeply rooted inhibitions. I am suggesting that the smallness of the effect may have been an essential part of the learning. Their inhibitions against "things technical" had been so strong that they had grabbed the LEGO® pieces when no one was looking . . . you might even say, behind their own backs. I am convinced that the process of appropriation would not have worked if they had been **told** to put lights in their house. The essence of the experience was in its delicacy.

The story is not really about girls. Many people would like to "appropriate the technical" but have not found a route that works well with their sense of themselves, one that is syntonic. Nor is this story only about gaining access to the technical. Many people have similar inhibitions about painting or composing music. Some have inhibitions about learning anything.

I find the story moving because it highlights the personal nature of all significant learning. I have fallen in love with LEGO® TC logo because it offers so many such opportunities for teachers and students to work together to build their learning environment.

* * * * * * * *

I have chosen in this preface to talk mostly about silent teaching — about the times it is empowering to be quiet; about kinds of knowing that cannot be put into words. Don't get me wrong. I have no intention of slighting the other parts of the teacher's job. I am certainly not saying that teachers should not teach. The four girls in the story might never have built their house without the support and technical help of an instructor. But the instructor had the wisdom to know what could be taught and what had to be learned. Showing the girls principles of construction was empowering — as was taking seriously their project of building a house when other children were busy with other high-tech projects. It might have been profoundly disempowering to ask, even in the gentlest voice, "Why don't you build a truck with a motor?" or even, "What about putting lights in your house?"

I hope no one will read me as suggesting that all LEGO® TC logo learning has the gossamer quality of delicate shifts of self-image. There is much more. Shifts of self-image are delicate only in their beginnings: within a few weeks the four girls were quite robustly into technical work. Thus, the girls acquired a wide range of skills and facts — the same skills and facts that one finds in the most codified educational objectives. LEGO® TC logo provides a learning environment especially well-suited to acquiring a sense of scientific method and engineering design as well as some centrally important specific knowledge.

It is not necessary for me to repeat here what is done so well in the rest of the *Teacher's Guide*. There you will find information about using LEGO® TC logo and suggestions on how to link LEGO® TC logo activities with traditional curriculum content. But I would like to add one more thought about "silent" and "talkative" teaching.

Consider the concepts of fractions, ratio and proportions. These concepts come up very frequently in LEGO® TC logo work. They come up in questions about scaling models. They come up in the uses of pulleys and gears. They come up in programming distance, rate and time procedures. You might feel that this is a wonderful chance to talk about those concepts in a motivated context. My advice is to restrain yourself. During LEGO® TC logo classes, give students freedom to explore gears and pulleys and programming. Then, when you discuss fractions and ratios in the classroom, draw upon the LEGO® TC logo experience.

I would like to end by stressing the importance of "culture building." The true value of LEGO® TC logo, like the value of Logo in general, is realized when it becomes part of the culture of the school and of the home. By this I mean it becomes a shared experience. LEGO® TC logo, like all learning activities, is most powerful when it becomes part of the pool of ideas, images and metaphors everyone uses to communicate and to think.

Chapter 1

What is LEGO® TC logo?

LEGO® TC logo is many things. It is a construction set. It is a programming language. It is a science activity. It is a math activity.

Most of all, it is an **environment for learning.**

Consider these scenes from an elementary school in Boston:

* It is a hot summer day. So a 10-year-old boy builds a LEGO® fan and writes a LEGO® TC logo program to turn the fan on and off.

* A 12-year-old boy builds an oven, using LEGO® lights to represent the heating elements. He writes LEGO® TC logo programs for different "recipes." Other students see the oven and begin building other household machines. One girl builds a pop-up toaster. Another builds a programmable sewing machine.

* Three fourth-graders, two boys and one girl, decide to build an elevator. While one boy builds the outside structure, the other works on the gear mechanism. The girl writes a LEGO® TC logo program to control the elevator. Once the elevator is working, the children add a touch sensor at the top. Then, they change the program so that the elevator automatically reverses direction when it reaches the top floor. And it works!

In each of these projects, children are acting as **inventors**. LEGO® TC logo creates an environment in which children **want** to create, invent, and explore.

** What do students learn?

LEGO® TC logo is rich with learning opportunities for both elementary and junior high students. Of course, students will learn about **computers**. By using the computer to control their inventions, students will learn the fundamentals of computer programming and they will gain an understanding of how computers are used in the world.

But LEGO® TC logo is not just a computer curriculum. Rather, it cuts across many disciplines. LEGO® TC logo offers a new way for students to learn important ideas in math and science, in social studies and engineering, in architecture and design.

As an example, think about the way **science** is taught in most schools. In most science lessons, students learn a set of facts and definitions. Even in "hands-on" science lessons, students are usually told exactly what to do. They are re-creating someone else's experiment.

LEGO® TC logo offers a different approach. In LEGO® TC logo, students act like real scientists. They **do** science. They come up with ideas, run experiments, and revise their plans based on the results. Students learn both the **concepts of science** (like friction, motion, and mechanical advantage) and the **process of science**. More broadly, students learn important **problem-solving** and **critical-thinking** skills.

Consider a student who builds a LEGO® car then writes a LEGO® TC logo program to make the car move. The car works fine on a flat table, but it is unable to "climb" a steep hill. How can the student solve this problem? One approach is to change the gears on the car. Through this activity, the student can gain first-hand experience with **gear ratios** and **mechanical advantage**.

Of course, students also learn important ideas in other curriculum areas. Each project in the student booklets highlights ideas in **math** (such as measurement and fractions) and **social studies** (such as invention and transportation), as well as **science**. The charts on pages 20-22 list some of these curriculum links.

Equally important, students learn **personal skills** while working on LEGO® TC logo projects. For example, students are likely to learn the value of **collaboration** and **teamwork**. In many cases, the spirit of cooperation developed during LEGO® TC logo activities carries over to other classroom activities.

LEGO® TC logo also helps students gain a new sense of **self-confidence**. Students working with LEGO® TC logo almost always end up building things that are better than they expected. This is particularly important for students who have never shown a strong interest in school activities. LEGO® TC logo can help these students develop a new attitude about school in general.

Chapter 2

Setting Up

You can setup LEGO® TC logo wherever you have (1) access to a computer and (2) space for students to build and test their projects. Your own classroom is an ideal setting. The school computer lab is also a possibility.

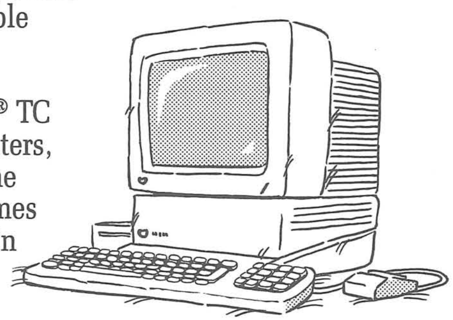
Wherever you setup, it is a good idea to arrange clusters of desks or tables so that students can work together and share ideas.

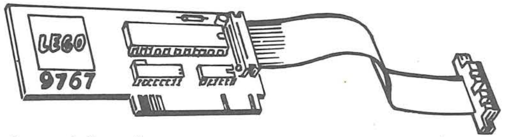
Here's what you need to get started:

** A personal computer

The LEGO® TC logo system will work with the Apple® IIe™ and the Apple IIGS™ computers, and with most IBM® and IBM-compatible computers.

You need only one computer for LEGO® TC logo activities. If you have more computers, students will be able to spend more time programming their projects. This becomes increasingly important as students begin working on more advanced projects.

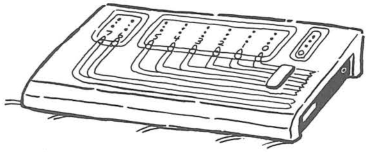




** Interface card

The LEGO® interface card enables the computer to communicate with the "outside world." You need one interface card (pictured above) for your Apple IIe or Apple IIGS computer. Or you need one interface card (Item 9771) for your IBM or IBM-compatible computer.

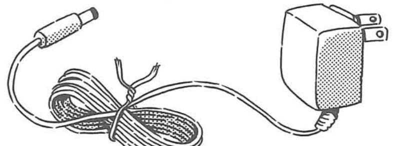
To install your interface card, follow the appropriate directions in the *Setup Guide*.



** Interface box

The LEGO interface box acts as a "middle man" between the computer and the LEGO models. The box has 12 "ports" where you can plug LEGO wires. Through these ports, you can send messages to LEGO motors and lights, or receive messages from LEGO sensors.

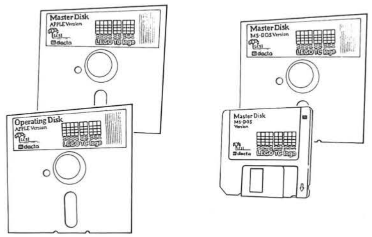
** Transformer



The transformer provides electrical power to the interface box to control LEGO motors and lights.

** LEGO® TC logo disks

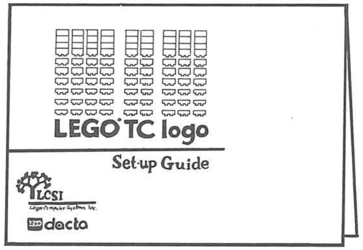
The LEGO® TC logo disks include the **software** for LEGO® TC logo activities. Once this software is loaded into the computer, you can write commands and programs to control the machines you build out of LEGO pieces.



The LEGO® TC logo software also includes graphics and text-processing capabilities, so that students can draw pictures and write stories about their LEGO® TC logo projects.

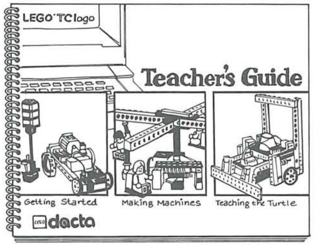
** *Setup Guide*

The *Setup Guide* explains how to install the LEGO interface card in your computer and how to begin using the LEGO® TC logo software.



** *Teacher's Guide*

You're reading it! The *Teacher's Guide* is divided into two parts.

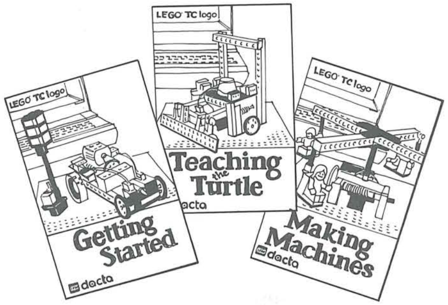


The first five chapters present an **introduction** to LEGO® TC logo. After reading these chapters, you'll be ready to start using LEGO® TC logo with your class. This chapter surveys all the materials and explains how to set them up. Chapters 3 and 4 give suggestions on how to use LEGO® TC logo with your class. Chapter 5 describes a quick, hands-on project that will help you become familiar with LEGO® TC logo.

Chapters 6 through 8 provide **supplementary notes** for each of the three student booklets. These notes suggest topics for discussion, further activities, and links to standard curriculum areas.

** Student booklets

Each of the three student booklets serves a special purpose:

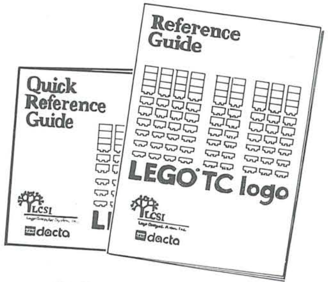


Getting Started describes several simple projects involving cars and traffic lights. By working on these projects, students can learn all of the basic commands needed to control the motors, lights, and sensors in LEGO® TC logo. This is a good place for students to start.

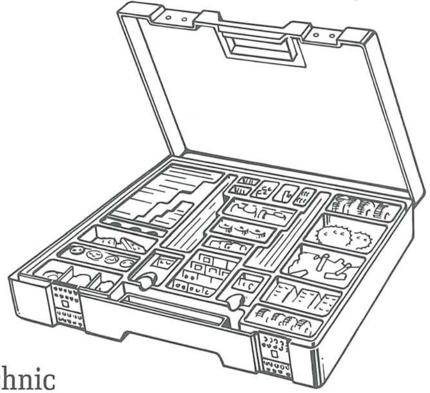
Making Machines describes how to build and program three projects: A merry-go-round, a washing machine, and a conveyor belt. These projects further develop many of the basic ideas introduced in the *Getting Started* booklet, while also introducing new programming concepts. By working on these projects, students can explore how machines affect our lives at home, at work, and at play.

Teaching the Turtle explains how to program a LEGO® "turtle" (a simple programmable robot). The turtle has proven to be a very effective device for introducing children to programming. At the same time, the LEGO turtle can also be used in very sophisticated programming projects.

** *Reference and Quick Reference Guides*



The *Reference Guide* includes a complete listing and description of all of the commands used in the LEGO® TC logo software. The *Quick Reference Guide* summarizes the most frequently used commands.

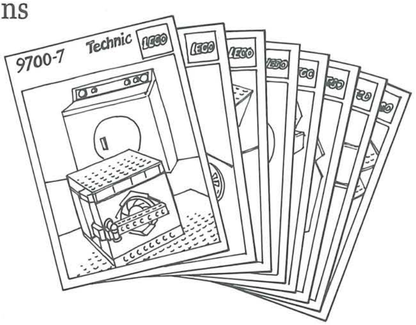


** Technic Control 0 building materials

The LEGO building materials include bricks, beams, gears, axles, wheels —and lots more. In all, there are more than 450 different pieces in the Technic Control 0 set. Each set includes enough pieces for about 4 students.

** Activity cards

The Activity cards provide instructions for building these LEGO models:



1. Building hints
2. Traffic Light
3. Motorized Car
4. Starting Gate and Finish Line
5. Conveyor Belt
6. Merry-Go-Round
7. Washing Machine
8. Turtle

** Resource Table

You might want to set up a LEGO® TC logo **resource table** in the classroom. The table could include books about machines, technology, computers, motion, and inventors. Some suggested books are listed in the Bibliography.

The table could include magazine articles about science and technology. You could also display some "real" machines (egg beaters, screwdrivers, etc.). Students could use these machines as models for their own building.

** Inventor's Notebook

It is a good idea for each student to keep an "Inventor's Notebook." Students can use these notebooks for many things.

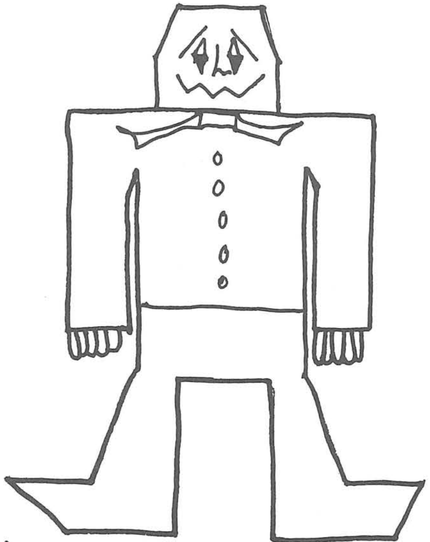
- They can write down or draw ideas for projects.
- They can note which ideas work—and which ones don't.
- They can keep data on experiments they run.
- They can make drawings of projects at various stages.
- They can make "LEGO® TC logo Patents" using the worksheet provided on page 147 of the *Teacher's Guide.* These patents can be included in their notebooks.

Any plain paper notebook will work fine as an Inventor's Notebook. Another option: students could keep notes directly on the computer, using the text-processing and graphics capabilities of the LEGO® TC logo software.

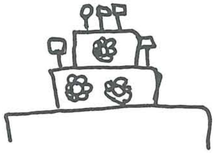
Following are examples of how students might use their Inventor's Notebooks.

My Robot

My Robot is like a human being. Some people think he is one but he is not. He is programmed to do what I say. I programmed him to clean the house. I wish that you saw what he did. He made all the wrong things I told him to sweep but he heard the wrong thing and ate all the sweets. He made a mess all over the house!



Diego Munoz
L.W. Batchelder School
Grade 4



```
To Candle
  Green
  Red
  Yellow
  End
```

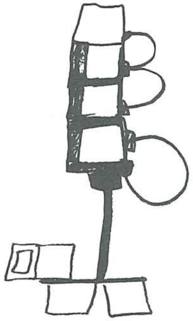
```
To Green
 Tto 3 onfor 20
   on
 End
```

```
To Red
 TTo 1 20
 Onfor 110
   on
   End
```

```
To Yellow
 Tto 2
  onfor 20
   On
   End
```

1. Check the screen and make sure you spelled everything right.
2. Check the lights and interface box.
3. Check the bulb.
4. Check the wires.
5. Check loose wires.
6. Check the sensor.

Stop light



Today we tested the stoplight agian. Look at the light on the bottom. It didn't light up because it wasn't connected. Today we connected it and it lit up.

```
TO Yellow-1

Talkto 2
Onfor 10
end
```

Andrea Martin
grade 4
Batchelder School

Chapter 3

Focusing on Projects

LEGO® TC logo is a **project-oriented** activity. A LEGO® TC logo classroom resembles a workshop, or perhaps a beehive. Groups of students work together on projects. They start by building. Then testing and changing. Then programming. Then perhaps more building or changing.

Through it all, groups can interact with one another, spreading new ideas through the classroom.

There are several advantages to the project-oriented approach:

First, projects give students an understanding of **process**: The process of design, the process of scientific investigation, the process of invention. These ideas are ignored in most school lessons.

At the same time, projects provide **context** and **motivation**. Students confront new ideas as part of an activity that they find interesting and meaningful. As a result, students are more likely to learn from their experiences.

** Guided Projects vs. Personal Projects

There are two types of LEGO® TC logo projects. The students booklets provide a series of **guided projects**. Through these projects, students can learn the "tricks of the LEGO® TC logo trade." The projects introduce students to a broad array of skills and ideas, from science to social studies to programming.

Alternatively, students can create **personal projects** based on their own ideas. These projects can be simple extensions or modifications of the guided projects. Or they can be totally new inventions.

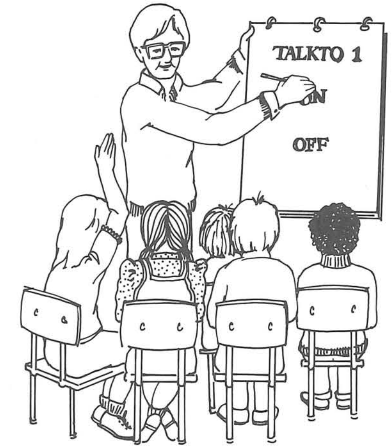
These open-ended projects play an important role in the learning process. Students often feel a greater sense of involvement and accomplishment when they work on personal projects.

** The Role of the Teacher

In project-oriented activities, it is important to give students the opportunity to explore and discover on their own. But it is also important for you to provide some structure to the discovery process.

You can support the discovery process in several ways. You can act as a:

Catalyst. You can introduce new ideas and provoke students to explore in new directions. For example, you might use the first ten minutes of each LEGO® TC logo session to introduce new programming concepts and commands. Students could then integrate the new ideas into their projects. Alternatively, you might introduce new ideas to students individually and let them share the ideas with their classmates.



Discussion Leader. You can start discussions on ideas related to the LEGO® TC logo activities. For example, you might discuss the differences between animal senses and machine sensors. You could initiate these discussions before, during, or after the LEGO® TC logo sessions. Through these discussions, you can relate LEGO® TC logo experiences to curriculum topics in math, science, social studies, or writing. Chapters 6 through 8 suggest possible topics for discussion.

Consultant. Students will undoubtedly encounter "bugs" (things that don't work as expected) while building or programming. "Debugging" is an important part of the learning process. By fixing the bugs, students will gain a better understanding of **why** the bugs occurred. In the process, students will develop stronger **problem-solving skills**.

You can assist in this debugging process. You might ask a series of questions like this:

"Well, what seems to be the problem?"

"How have you tried to fix it?"

"Let's try to figure out **why** it's not working."

"Now that we know **why** it's not working, how can we fix it? Let's brainstorm a bit. What are some possible solutions?"

"OK, which ideas should we try first?"

** Evaluating Student Performance

It is often misleading to judge students' learning experiences in LEGO® TC logo on the basis of their finished projects. Instead, you should focus on the **process**, not the **results**, of their work. A student's level of involvement and willingness to try new things are generally more important than the quality of the student's finished project.

Of course, it is most useful to evaluate students on an **individual basis**. Rather than comparing a student with others in the class, you should look at the student's individual progress during LEGO® TC logo activities.

In evaluating students, you might consider the following categories:

Group work
Does the student work well with other students?
Does the student share ideas and materials with others?

Individual work
Does the student have the discipline to work individually?
Does the student become frustrated when things don't work right away?

Divergent thinking
Does the student come up with creative ideas for projects?
Does the student use "errors" as a source of new ideas?

Convergent thinking
Does the student follow through and finish projects?
Does the student explore ideas in depth?

Hands-on
Does the student "experiment" with the LEGO® materials?
Does the student "experiment" with LEGO® TC logo programming?

Heads-in
Does the student reflect upon the meaning of things that happen?
Does the student understand "big ideas" like friction and gear ratios?

Chapter 4

Routes Through the Material

There are many ways to use LEGO® TC logo in your classroom. You can view LEGO® TC logo as a science activity, a computer activity, a math activity, or an interdisciplinary activity. You can use LEGO® TC logo with the entire class on a twice-a-week schedule, or you can set it up as one of several stations in an "open" classroom. You can use LEGO® TC logo for just a few weeks, or for the entire school year.

It is difficult to give estimates for LEGO® TC logo activities. One rule of thumb: students should probably have at least three classroom hours for working on each of the projects in the *Getting Started* booklet and at least four hours for each of the projects in the *Making Machines* booklet. Of course, students will need more time if they want to explore variations or extensions of the project.

In any event, it is important not to rush students through LEGO® TC logo activities. Students need time to explore, experiment, and think about the ideas they encounter in LEGO® TC logo.

The following charts suggest three possible "routes" for using LEGO® TC logo with your class. Each chart shows a sequence of projects, along with a list of advantages and disadvantages. Projects are indicated by the following symbols:

Getting Started Projects

Making Machines Projects

Teaching the Turtle Projects

Personal Projects

Route 1

Advantages

1. Fast, direct way to learn basic LEGO® TC logo skills
2. Orderly progression of new ideas

Disadvantages

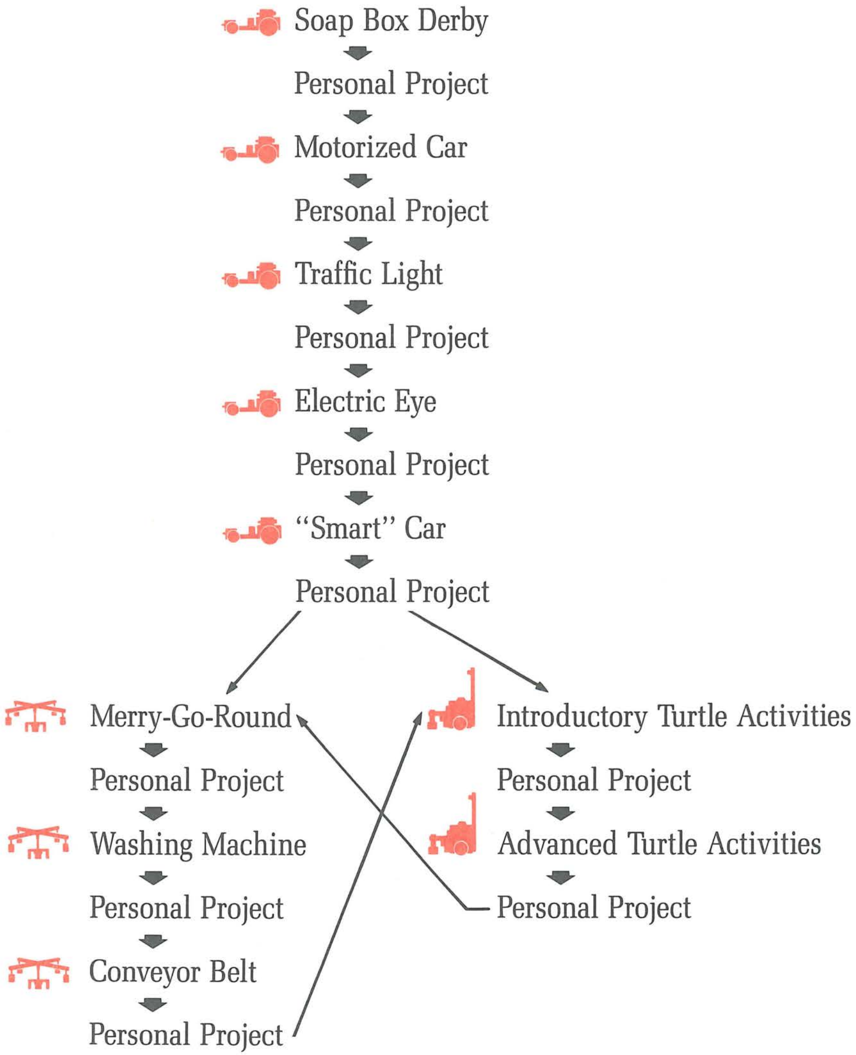
1. Limited opportunities for student creativity

Soap Box Derby
▼
Motorized Car
▼
Traffic Light
▼
Electric Eye
▼
"Smart" Car
▼
Merry-Go-Round
▼
Washing Machine
▼
Conveyor Belt
▼
Introductory Turtle Activities
▼
Advanced Turtle Activities
▼
Personal Projects

Route 2

Advantages
1. Fast, direct way to learn basic LEGO® TC logo skills
2. Orderly progression of new ideas
3. More freedom for students than Route 1

Disadvantages
1. Takes more time than Route 1



Route 3

Advantages
1. Greater opportunities for student creativity
2. Students might be more motivated

Disadvantages
1. Not enough structure for some students

Soap Box Derby
Motorized Car
Traffic Light
Personal Project *
Personal Project *
Personal Project *
Personal Projects continued *

* Activities in the student booklets would be used as a reference in each personal project.

** Links to the curriculum

Regardless of which route you choose, there are manys ways to link LEGO® TC logo projects to standard topics in your school curriculum.

The next three pages show three "curriculum charts," one for each of the student booklets. For each student activity, these charts show:

* New LEGO® TC logo primitives and concepts
* Science concepts
* Math concepts
* Social-studies ideas

Getting Started Curriculum Chart

Activity	LEGO® TC Logo Skills	New Logo Primitives	Science Concepts	Math Concepts	Social Studies Ideas
Soap Box Derby	Basic construction with LEGO® building elements Wheels, Axles		Simple machines Motion, Friction Scientific method Data Collection	Measurement Averaging Graphing Time-rate-distance	Invention Inventors

Motorized Car	LEGO® motors
	Gears, Pulleys
	Computer control
	Simple Logo commands
		`on`, `off`
		`onfor`
		`rd`, `cc`
		`talkto`
			Forces
			Motors, Electricity
			Gears
			Mechanical advantage
			Programming
				Time-rate-distance	History of machines
					Transportation
					History of the computer
					Uses of computers

Traffic Light	Logo lists
	Logo procedures
	Recursion
	Touch sensors
	Logo conditionals
		`repeat`
		`wait`
		`alloff`
		`listento`, `show`
		`sensor?`, `not`
		`waituntil`
		`if`, `ifelse`
			Light
			Color
			Animal senses
			Machine sensors
				Codes
				Binary arithmetic
				Recursion
					Communication

Electric eye	Optosensors Computer timer	`timer` `resett`	Vision Time Clocks		History of clocks
"Smart" car	Gear trains Sensors on machines Counter Procedures with inputs	`counter` `resetc`	Gears Mechanical advantage Calibration Measurement tools	Fractions Ratios Measurement Time-rate-distance	History of machines Transportation Maps

Making Machines Curriculum Chart

Activity	LEGO® TC logo Primitives and Concepts*	Science Concepts	Math Concepts	Social-Studies Ideas

Merry-Go-Round	`readchar`
	`name`
	`tone`
	naming/variables*
		Circular motion
		Centrifugal force
			Circles
			Fractions
				History of entertainment

Washing Machine	`or` `output`	Time/clocks Using sensors for safety	Logic Time Codes	Automated house House of the Future History of household chores Product safety
Conveyor Belt		Intelligent machines Robots	Measurement	Automation Industrial Revolution Factory of the Future

Teaching the Turtle Curriculum Chart

Activity	LEGO® TC logo Primitives	Science Concepts	Math Concepts	Social-Studies Ideas

Introductory Activities	`tfd`
	`tbk`
	`tlt`
	`trt`
		Motion
		Time-rate-distance
		Robots
			Measurement
			Angles
			Ratios
			Geometry
				Machines vs. animals
				Animal behavior

Advanced Activities	`gofd`
	`gobk`
	`golt`
	`gort`
	`offturtle`
	`seteven`
	`setodd`
		Intelligent machines
		Animal senses vs. machine sensors
		Simulation
			Calibration	

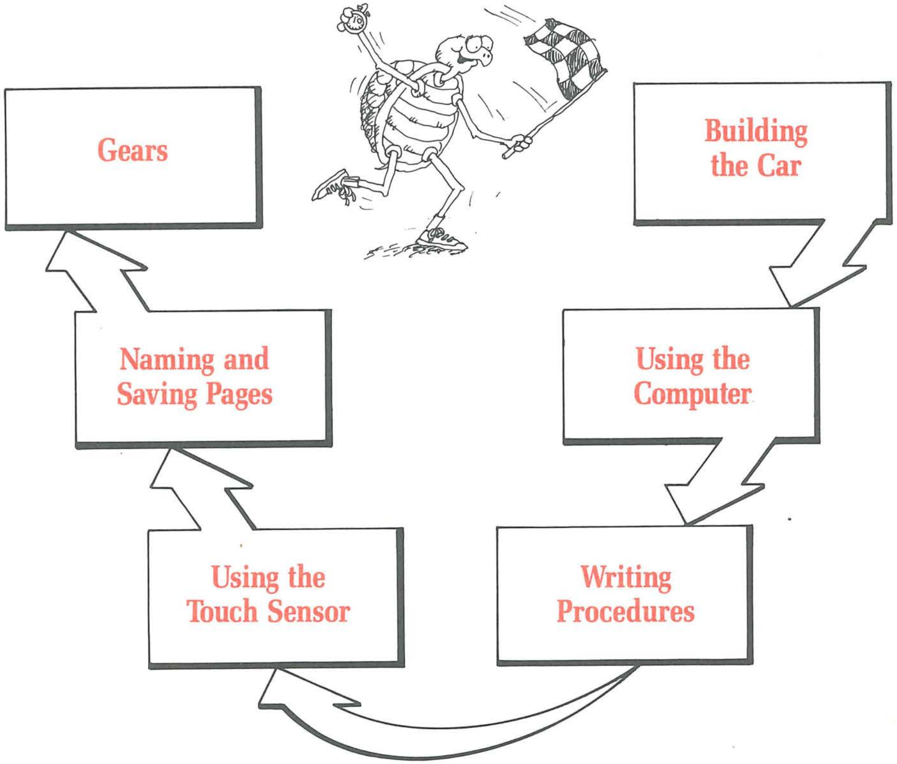
Chapter 5

Getting Familiar with LEGO® TC logo

The best way to become familiar with LEGO® TC logo is to do it yourself.

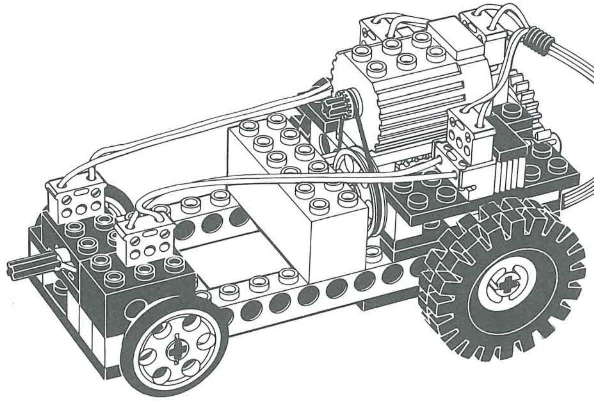
This chapter presents a "guided project" to help you get started. The project involves building and programming a motorized LEGO® car. By working on this project, you can gain experience with both the LEGO® construction system and the Logo programming system. Most importantly, you will experience the pleasures (and problems) that your students will encounter when they begin working with LEGO® TC logo.

This project combines ideas from several of the projects presented in the students' *Getting Started* booklet.



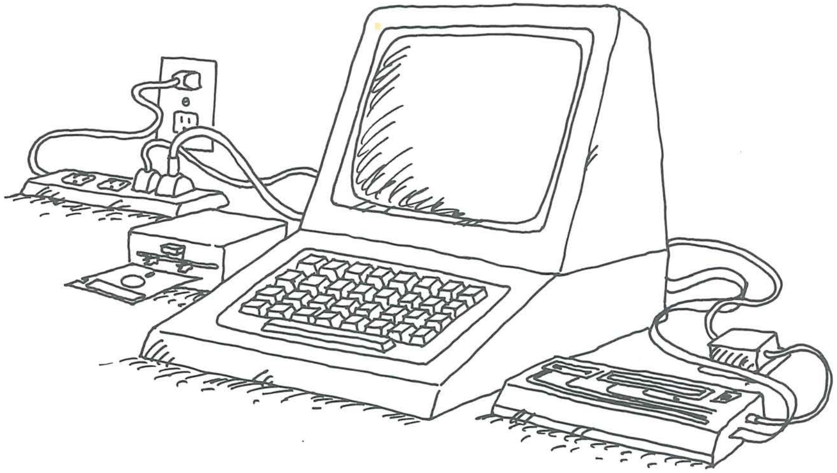
Building the car

Activity Card 3 shows how to build the car. The LEGO® Activity Cards are excellent examples of **graphic communication**. The pictures provide all of the information you need. Go ahead and build the motorized car.



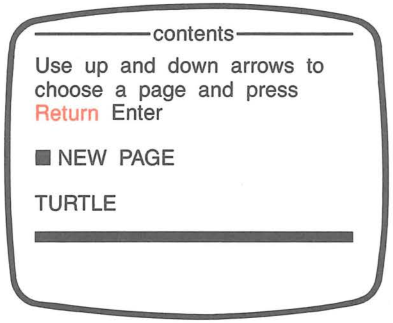
Using the Computer

If you have not yet installed the interface card in your computer, do so now, following the instructions for your computer in the *Setup Guide*. Then connect the interface box and transformer, as described in the *Setup Guide*.



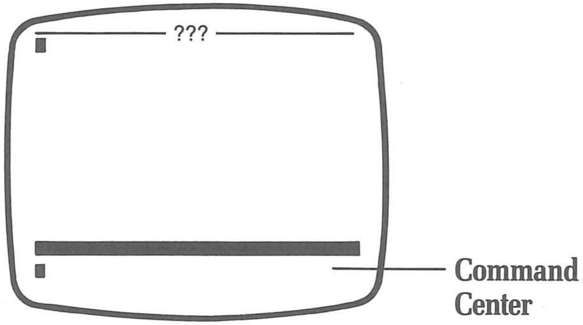
The *Setup Guide* also explains how to start the LEGO® TC logo software on your computer.

When the software is finished loading, you will see `Contents.` The blinking light is called the **cursor.** You can move the cursor by pressing the **up** and **down** arrow keys. Try it. Place the cursor over `NEW PAGE` and press the Return Enter key.



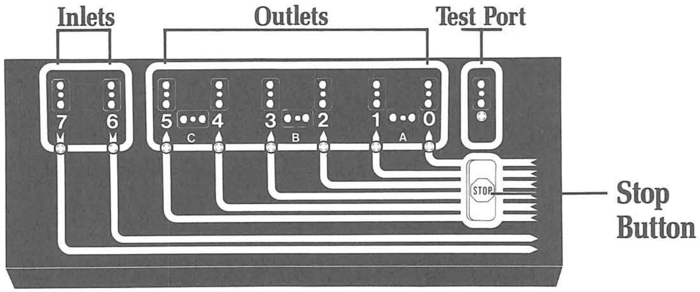
```
contents
Use up and down arrows to
choose a page and press
Return Enter

■ NEW PAGE

TURTLE
```

A new page will appear. The bottom part of the screen (below the solid line) is called the **Command Center**. This is where you type instructions to the computer.



```
???
```

To control the car, you must connect the car motor to the **interface box**.

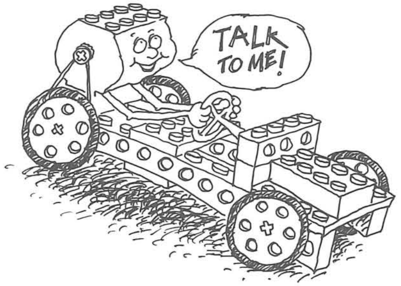


The interface box has 12 ports. The port above the red stop button is called the "test port." The test port supplies power at all times. Plug one end of a long LEGO® wire into the test port and plug the other end into the motor of your car. The car should begin moving. If the car does not run smoothly, look for pieces that are slipping and/or rubbing against one another.

To control the car from the computer, unplug the LEGO® wire from the test port and plug it into port A. (You can connect wires together, like extension cords, if you want a longer wire.)

To begin, type `talkto "A` in the Command Center. Make sure to include the quotes before (but not after) the letter `A`. Also, make sure to leave a space between `talkto` and `"A`, but do **not** leave a space between `talk` and `to`. Now press the Return key. On an IBM computer, press the Enter key – shown as ↵ on the keyboard. This command tells the computer that it should "talk to" port A on the interface box.

Apple users: If your program won't run or it "hangs" so you can't change it, check that the slot card is in slot 3 or change the slot card setting. (See *Setup Guide* for details.)



Now type `onfor 20` in the Command Center. (Make sure to leave a space between `onfor` and `20`, but do **not** leave a space between `on` and `for`.) Press the Return Enter key. The car will move for two seconds.

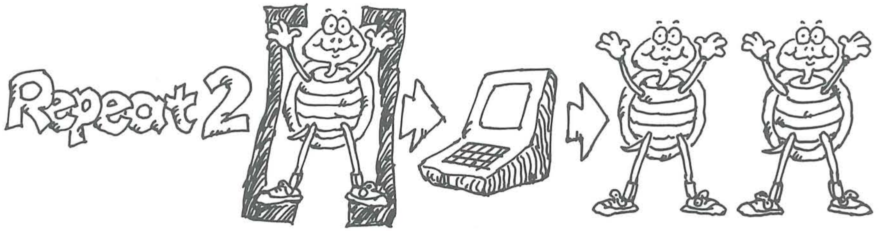
Next, type `on` and press Return Enter. The motor will turn on and stay on.

To make the car reverse direction, type `rd` and press Return Enter. The car will start moving in the opposite direction.

To turn the motor off, type `off` and press Return Enter.

You can put several commands on one line. For example, type `onfor 20 rd onfor 20` and press Return Enter. The car will go forward and then back (or back and then forward!).

You can also combine these commands with standard Logo commands. For example, type `repeat 4 [onfor 15 rd]` and press Return Enter. (Make sure to use **square** brackets.) This tells the computer to repeat the list `[onfor 15 rd]` four times.



You can use other ports if you like. Unplug the wire from port A and plug it into port B. To make the computer "talk to" port B (instead of port A), type `talkto "B` and press Return Enter. Now if you type `onfor 20` (and press Return Enter) the motor plugged into port B will go on for two seconds.

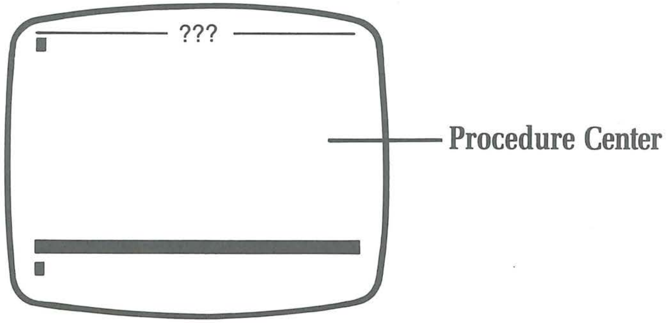
A Note About "Clocks": The LEGO® interface card contains the clock that LEGO® TC logo uses when you set *time values*. Also, if you want to program music using the `tone` primitive, LEGO® TC logo will use that clock to create the correct pitch or note. This means you must have the LEGO interface card in your computer if you want to use programs with time values or musical notes.

Writing Procedures

You can also create new commands of your own. Let's say you want the car to go out and back. You can do this by typing `onfor 20 rd onfor 20` in the Command Center. But wouldn't it be nice if you could simply type the word `car`?

To do that, you must create a **procedure** for `car`. That is, you must tell the computer exactly what `car` means. Creating new procedures is an important part of Logo programming. Once you define a new word (like `car`), you can **save** the procedure to use again later. You can also use the procedure as a building block to create even more complex procedures.

To define a new procedure, you must move the cursor to the top part of the computer screen (above the solid line). This is called the Procedure Center. If you are using an Apple computer, to move the cursor **up**, hold down the Apple key () and press the **U** key (without letting go of the key). If you are working on an IBM computer, hold the Control (**Ctrl**) key down and press the **U** key. The cursor should jump to the top of the page.



Now type in the definition for `car`:

```
to car      ◀ Procedures must start with to
onfor 20
rd
onfor 20
end         ◀ Procedures must end with end
```

To run the new procedure, you must return to the Command Center. You move the cursor **down** to the Command Center by holding down the Ctrl key and pressing the **D** key.

Now type `car` and press Return Enter. The car will go for two seconds, reverse direction, then come back to where it started.

One possible bug: The computer might be "talking to" the wrong port. If the car motor is plugged into port A, type `talkto "A` (and press Return Enter) to make sure that the computer is talking to port A.

Using the Touch Sensor

Create a touch sensor by inserting a short axle into a touch-sensor brick. Now attach the touch sensor to the front of the car, as shown on Activity Card 3. Plug one end of a LEGO® wire into the sensor and plug the other end into port 7.

The touch sensor sends messages to the computer. When anything (or anyone) presses the touch sensor, the sensor reports `true` (and the green light next to port 7 goes on). When nothing is pressing the sensor, the sensor reports `false` (and the green light is off).

By “listening to” the touch sensor, the computer can “know” when your car bumps into a wall. For example, you can write a procedure that makes the car reverse direction when it bumps into a wall. First move the cursor up to the top part of the screen (press -U or Ctrl-U), then type this procedure:

```
to bump
talkto "A
listento 7
on
waituntil [sensor?]
rd
onfor 20
end
```

The `bump` procedure turns on the car motor and waits until the touch sensor reports `true` (that is, it waits until the sensor bumps into something). Then the computer reverses the direction of the motor and turns it on for 2 seconds.

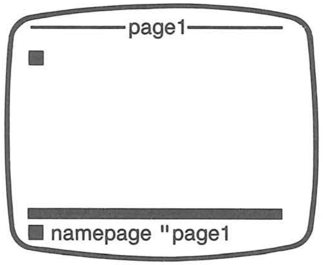
To run `bump`, first make sure the motor is plugged into port A and the touch sensor into port 7. Then move the cursor down to the Command Center (press -D or Ctrl-D). Type `bump` and press Return Enter.

One possible bug: the car might go backwards! If that happens, press -S or Ctrl-S to **stop** the procedure. Then type `off`. This command turns off the motor if it is still running. Now type `rd` to reverse the direction of the motor, and you are ready to try the `bump` procedure again.

Naming and Saving

You can save your page onto your disk. That way, you can use your procedures again later.

Before saving, you must name your page. Say you want to name the page `page1`. (Note: The name must be a single word.) Then type `namepage "page1` in the Command Center, and press Return Enter. Make sure to include the quotes before (but not after) `page1`. The new name should appear at the top of the page.



```
page1
namepage "page1
```

Next, press the Escape (**Esc**) key. That takes you back to the `Contents` page and automatically saves the `page1` page. Notice that `page1` now appears in the `Contents` list.

Gears

Before stopping, examine the **gear train** in your car. Gear trains can change the **speed** and **strength** of the machines you build.

The gear train in this car is designed to make the car **slower** and **stronger**. This is common in LEGO® design. LEGO® motors are very fast and very weak. You will often need to use gears to make your machines slower and stronger.

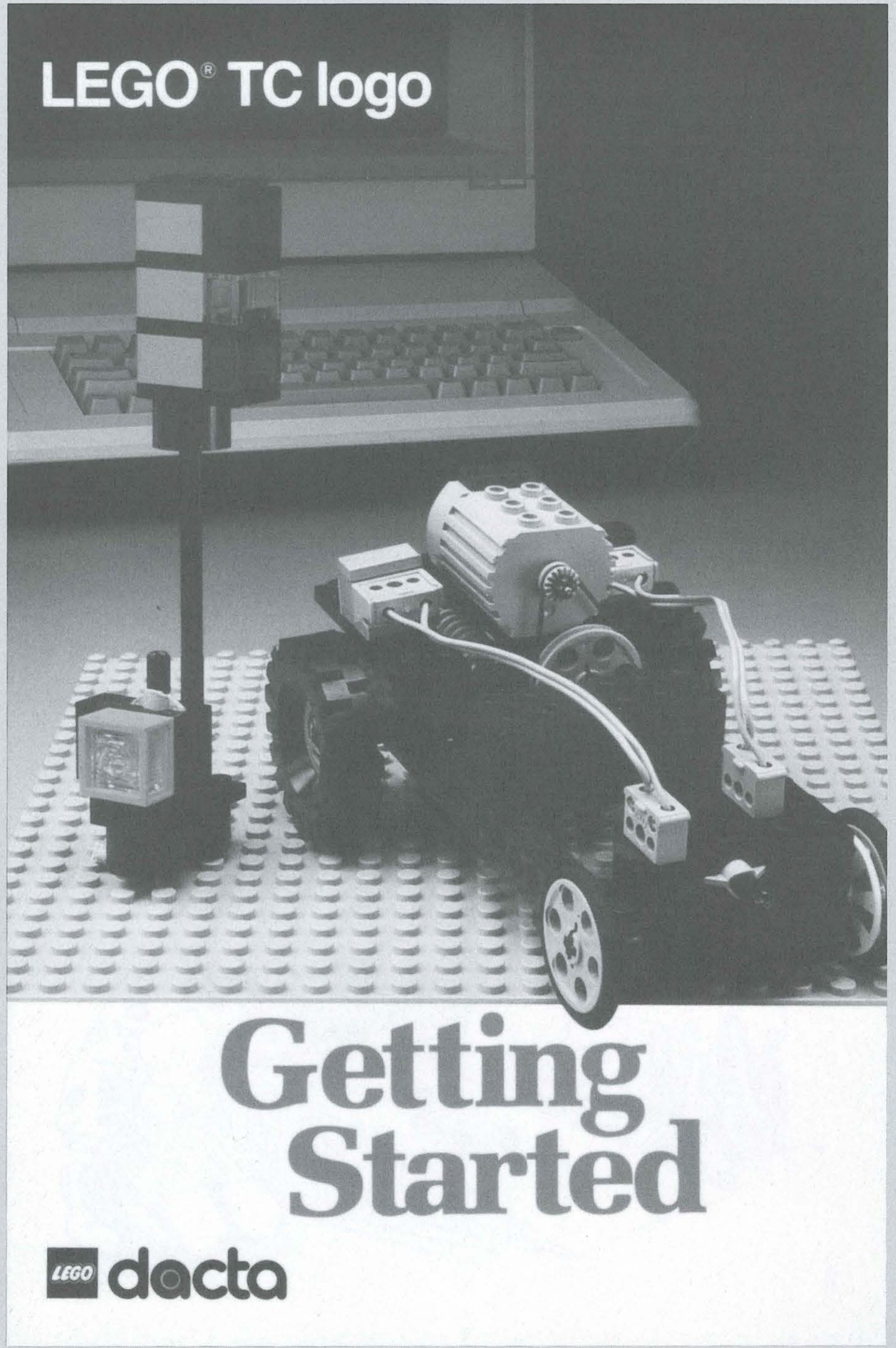
Take off the rubber band that connects the pulley on the motor and the pulley on the axle. Now look at the 8-tooth gear and 24-tooth gear on the back of your car. Turn the 8-tooth gear slowly and watch the 24-tooth gear turn. How many times must the 8-tooth gear turn to make the 24-tooth gear turn once? (Three. To keep track of the 24-tooth gear, you might mark one tooth with a magic marker.) So the 8-tooth gear is **three times faster** than the 24-tooth gear. But there is a trade-off. The 24-tooth gear is **three times stronger**. By giving up speed you gain strength.

Now turn the 24-tooth gear and watch the worm gear and the other 24-tooth gear underneath the car revolve. When you turn the first 24-tooth gear one revolution the worm gear turns once, causing the other 24-tooth gear to advance one tooth. So first 24-tooth gear and the worm gear must turn 24 times to make the second 24-tooth gear revolve once! Try it!

Now consider the motor and the pulley. The motor must turn several times to make the pulley turn once. (Of course, the rubber band must be back on!) So the pulley is slower than the motor, but it can exert a greater force.

Conclusion

You are now ready to run a LEGO® TC logo class. The sample project covered most of the basic concepts of the LEGO® TC logo system: of course, there is a lot more to learn. Before starting with your class, you might want to work through Project 5 in the *Getting Started* booklet (page 40). This project uses the car you have already built. By working through the suggested activities, you can learn about the LEGO® optosensor and the LEGO® counting wheel.



LEGO® TC logo

Getting Started

LEGO dacta

Chapter 6

Teacher's Notes for *Getting Started*

The *Getting Started* booklet is an introduction to LEGO® TC logo. Through a series of simple projects, the booklet introduces all the basic skills and ideas needed to build and program with LEGO® TC logo. The booklet assumes no previous experience with either LEGO® building or computers.

The booklet is divided into five projects:

** Soap Box Derby (motorless car)
** Motorized car
** Traffic light
** Electric eye
** "Smart" car

LEGO® cars are used in many of these activities. Once students have built their motorized cars, they might leave them assembled so that they don't have to rebuild them for future activities.

Project 1: Soap Box Derby

Key Ideas

- Introduction to LEGO® pieces
- Inventors and invention
- Scientific process
- Techniques for measurement, data collection
- Key science concepts: Friction, gravity, motion

Students can start by building with LEGO bricks. It is useful for students to look at Activity Card 1. The diagrams on this card show various ways that LEGO pieces can be used. These examples give students a feel for the range of possibilities in building with LEGO bricks.

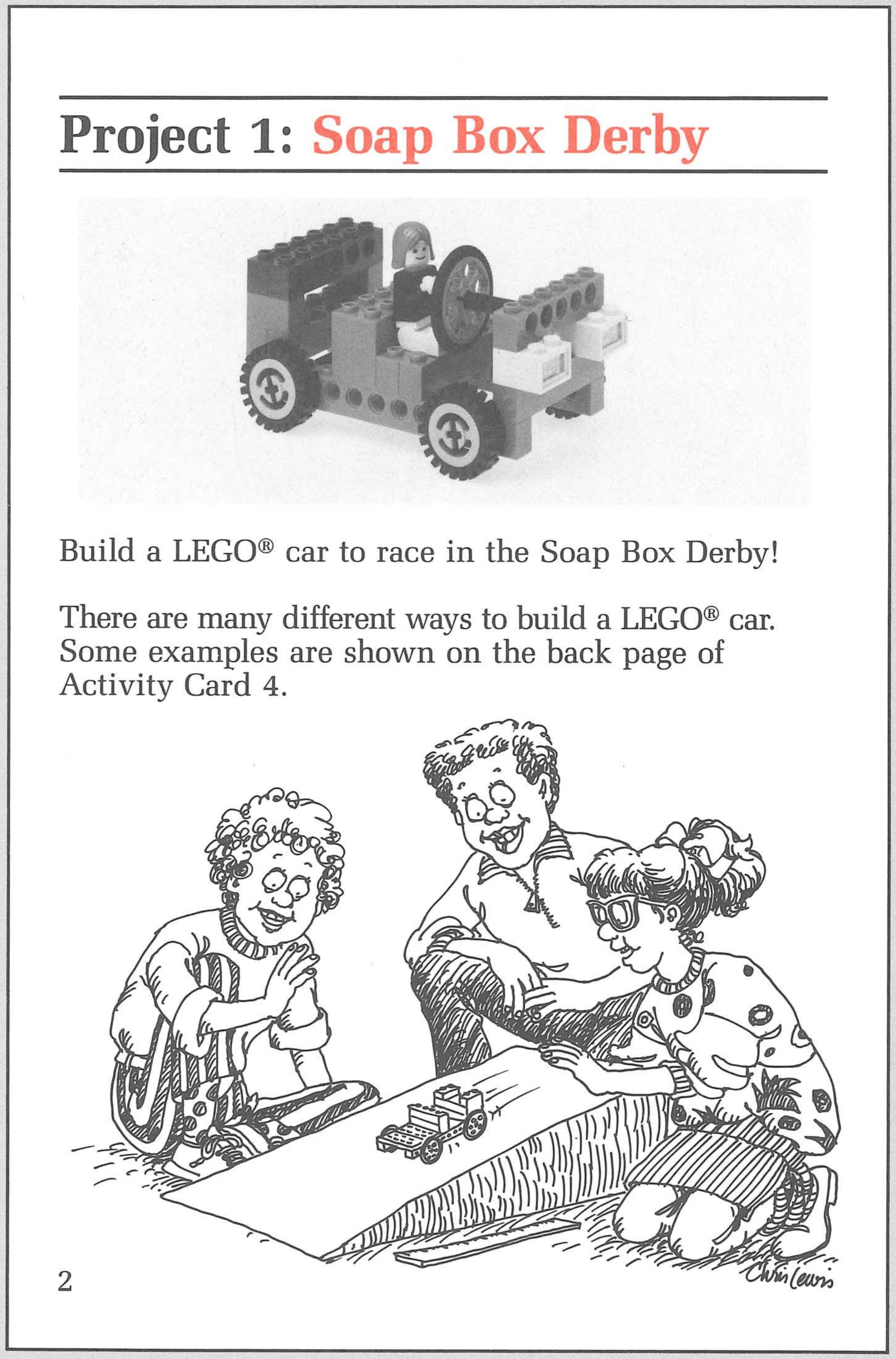
Students should view the car models on the back of Activity Card 4 as **examples.** They should feel free to develop their own designs. You might build a LEGO® car yourself ahead of time and show it to students as an example.

Use LEGO base plates (24 x 24 studs), sturdy cardboard or some other sturdy material for building the ramp. The ramp should be about one meter (or about three feet) long and 70 centimeters (or about two feet) wide. Raise one end about 30 centimeters (or about one foot) off the ground. Students can build an adjustable frame for their "base plate" ramp out of LEGO pieces!

Soap Box Derby. In introducing the activity, you might discuss "real-life" soap box races. For these races, children build cars (without motors) and race them down gentle hills. Each year, there is a national championship held in Akron, Ohio. The name "soap box derby" is a leftover from the days when soap was sold in large wooden boxes. Children built their cars out of these boxes. Today, children typically use other materials to build cars for the races, but the name "soap box derby" remains.

Worksheet. A worksheet is provided in Appendix 1 (page 145 of the *Teacher's Guide*) for you to copy and give to the students. This worksheet provides a chart for students to record the results of each car they race. The worksheet can be kept in each student's Inventor's Notebook.

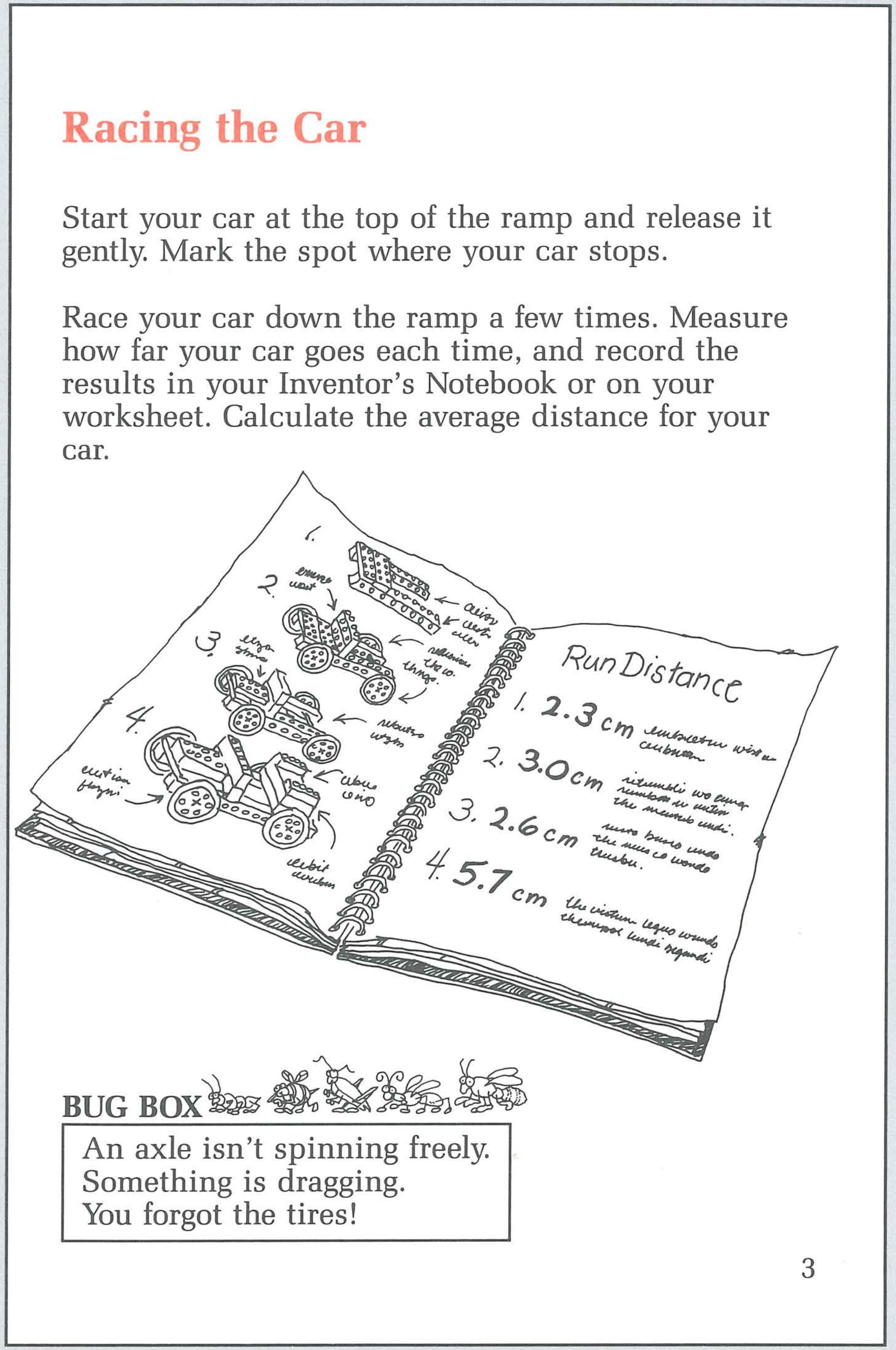
Invention. Ask the students to think about the design of their cars. Why did they build them in the way they did? How could they make their cars faster? Slower? Simpler? This is a good opportunity to discuss the **process of design.** Design always involves testing and changing, then more testing and changing.



Project 1: Soap Box Derby

Build a LEGO® car to race in the Soap Box Derby!

There are many different ways to build a LEGO® car. Some examples are shown on the back page of Activity Card 4.



Racing the Car

Start your car at the top of the ramp and release it gently. Mark the spot where your car stops.

Race your car down the ramp a few times. Measure how far your car goes each time, and record the results in your Inventor's Notebook or on your worksheet. Calculate the average distance for your car.

BUG BOX

An axle isn't spinning freely.
Something is dragging.
You forgot the tires!

Students can use masking tape to mark where the car stops (and then write their initials on the tape). That way, students can compare results from several runs.

Students should also record the distance their car travels on their worksheet. It is best for students to compete "against themselves" (comparing results from several runs), rather than competing against one another. That way, students can more easily determine which changes in design make a difference.

In addition to recording measurements, students can make drawings and explain their ideas either on their worksheet or in their Inventor's Notebooks. (See pages 14-15 of the *Teacher's Guide* for examples of Inventor's Notebooks.)

Measurement. To measure how far their cars went, students will need some type of measuring device. You can discuss different types of measuring devices and different types of measuring systems. Challenge 2 on page 5 of *Getting Started* (asking students to invent their own unit of measure) provides a good focus for this discussion. You can compare and discuss the different strategies/solutions that students develop.

Page 149 of the *Teacher's Guide* provides a copy of three different measuring devices: one in centimeters, one in inches, and one in "LEGO® units" (measuring the distance by the number of "studs" on a LEGO brick). The student books use metric measurements in most cases.

Bugs. Some of the cars will not roll smoothly at first. It is usually best for students to discover and fix the problems themselves. You can help by giving general tips on how to find problems. Some common bugs are:

Friction. Most problems with the cars result from friction on the axle or wheels. Friction is a force that opposes motion. If an object is at rest, friction makes it difficult to move the object. If an object is moving, friction slows it down or stops it. You might ask students: What is friction? What types of surfaces cause the most friction? How can you reduce friction? What problems are caused by friction? When is friction useful?

To reduce friction, students should turn their cars upside down and attempt to spin the wheels. If the wheels do not spin easily, students should try to figure out what is causing the friction. Most likely, one of the wheels is rubbing against another piece.

Sliding axle. In some cars, the axle will slide back and forth through the holes. This bug will not stop the car from moving, but it will make the forward motion somewhat erratic. This bug can be fixed by inserting the small gray spacers (called "bushings") along the axle.

The cycle of testing and changing, then more testing and changing, is an important part of the scientific process. This is often a slow gradual process. Students must have **time** to try out new ideas.

By recording data in an orderly way on their worksheet, students will better be able to determine which designs work best.

Scientific process. By modifying and testing their cars, students learn scientific ideas about motion in a very personal (and thus meaningful) way. For example, they might gain an intuitive feel for the concepts of speed, distance, time, acceleration, and friction. You can discuss how each of these concepts relates to the soap box race.

But equally important, students gain an appreciation for the scientific method. Students should make theories about their cars, then run experiments to test the theories. For example: If they add more weight, or more wheels, or both, does the car travel further? Students should keep track of their theories and experiments either in their Inventor's Notebooks or their worksheets.

Students can also learn the value of keeping a **control** in their experiments. If a student changes many parts of a car at once (say, by adding more weight and more wheels at the same time) he/she will have no idea which factor affected the car's performance. But if the student changes only one factor at a time, he/she will know whether that particular change was important.

Challenges

1. The steeper the ramp, the further the car should go. But if there is a lot of friction, the change in slope won't make much difference.

2. Students could use an arbitrary measuring standard, such as the length of someone's shoe. Maybe one car traveled five shoes, while another car traveled seven shoes. The key point: you need some type of standard to be able to compare measurements.

Making a Theory

There are many **factors** that could influence how far your car goes: The weight of the car, the number of wheels, the size of the wheels.

Which factors should you change to make the car go further? Should the car be heavier? Lighter? Should it have more wheels? Bigger wheels?

What is your **theory** of what makes the car go further?

Make a change and record the new results in your Inventor's Notebook. Change only one factor at a time. That way, you can tell which factors make a difference in the distance your car moves.

Challenges

1. What will happen if you make the ramp steeper? Raise the end of the ramp and try it!

2. Imagine that you didn't have any rulers in the classroom. How could you keep track of how far your car goes?

Note: You will be using your soap box car for other activities in this booklet so you may want to keep it assembled.

Simple machines. The inclined plane and the wheel-and-axle are two simple machines used in the soap box derby activity. Students will use other simple machines (pulleys, levers, screws) in later activities.

The wheel-and-axle is one of the oldest and most important inventions. You might discuss: Why do we need wheels? How did people move things before they invented wheels? You might also ask students to think of examples of inclined planes in the world around them (hills, wheelchair access ramps).

Data collection and analysis. Students need data to compare how their cars performed before and after design changes. Without good data, they have no systematic way of knowing if the changes improved performance. The worksheet from Appendix 1 provides students with a systematic way of recording data.

You might also discuss why it is a good idea to run an experiment several times. If a student runs his/her car down the ramp twice, it will not go exactly the same distance each time. You should discuss why this happens. Some possibilities: The car was pushed one time. Something came loose on the car. The car was released at an angle (not straight down the ramp).

This variation in the car's distance suggests the need for **statistical analysis**. The simplest type of analysis is **averaging**. Each time a student tests a new design, he/she should take several measurements, calculate the average and record it on his/her worksheet. You might discuss why the average is a better indicator of the car's performance than any one of the measurements. Also, you might discuss why the average becomes an even better indicator if you use five measurements to calculate the average, rather than just two or three.

Time-rate-distance. This is a natural time to discuss the relationship between time, speed, and distance. If students measure the time and distance for a particular race, they can calculate the **average speed** with the formula:
Average speed = Distance/Time.

The result is an **average** since the car is not moving at a constant speed during the race. It **accelerates** (due to gravity) while moving down the ramp, then **decelerates** (due to friction) while it is on the flat surface.

You can also talk about units of measurement. If students measure the distance in centimeters and time in seconds, the average speed will be in centimeters per second.

Another idea: Students could measure distances in two different units (say, inches and centimeters), and convert from one unit to another.

"Real" cars. Students could look through magazines and books to find pictures of different types of wheels or cars from different time periods.

Changing the goal. You could change the goal of the soap box race. For example, students might try to make their cars stop as close to the bottom of the ramp as possible. How should they re-design their cars? Another possibility: try to make the cars go for as much time as possible. Is this the same as trying to make the cars go as far as possible?

Using the computer. If the students have had some experience in Logo programming, you should encourage them to use the computer to perform the calculations they need during the soap box activity. In this way, the students will clearly see the power of the computer as a useful tool.

Here are two LEGO® TC logo procedures, one for calculating average speed, another for calculating the average of three numbers:

```
to speed :distance :time
show :distance / :time
end

to average :num1 :num2 :num3
show (:num1 + :num2 + :num3) / 3
end
```

For example, if you type `speed 80 10` and press the Return Enter key, the computer will print `8`. You might also discuss **units**. If the distance is in centimeters and the time is in seconds, the speed will be in centimeters per second.

If you type `average 6 10 17` and press the Return Enter key, the computer will print `11`.

Project 2: Motorized Car

Key Ideas

- Motors
- Transmission systems with gears and pulleys
- Science concepts: Force, electricity, mechanical advantage
- Introduction to the interface box
- Introduction to the computer
- Introduction to LEGO® TC logo software
- Computer control of motors
- New primitives: `talkto`, `onfor`, `on`, `off`, `rd`, `cc`.

Forces. This is a good opportunity to discuss forces. A force is a type of "push" or "pull." The force of gravity pulls objects toward the center of the Earth. Your hand can supply a force that pushes (or pulls) an object across the table. In a LEGO® car, the motor applies a force to a gear, which applies a force to another gear, which applies a force to an axle, which applies a force to the wheel, which applies a force to the ground. Thus, the car moves.

Friction is also a type of force. When an object moves in a certain direction, friction applies a force in the **opposite** direction. Thus, friction works to slow down the object.

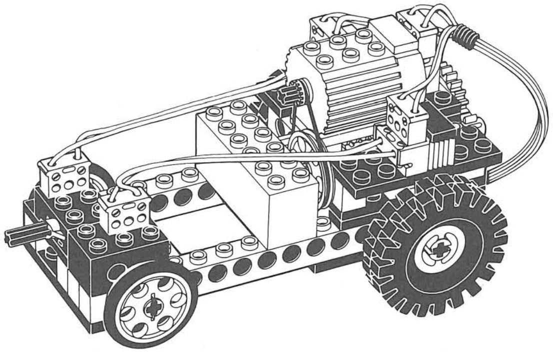
Speed/strength tradeoff. You can use gears to change the **speed** and the **strength** of a system. LEGO motors turn with great speed, but relatively little strength. By connecting small gears (or pulleys) to big ones, the system becomes stronger but slower. Alternatively, you can connect big gears (or pulleys) to small ones to increase speed at the expense of strength.

This **tradeoff** between strength and speed is a very important idea in designing mechanical systems. This topic is discussed in greater detail in Project 5 (page 40 of *Getting Started*).

Project 2: Motorized Car

To start a car moving, someone or something must supply a **force**. When the car rolls down a hill, the force comes from **gravity**. When you push the car with your hand, **you** supply a force.

Another way to supply a force is with a **motor**. Build the motorized car on Activity Card 3. You will use this model in activities throughout this book so keep the car assembled.



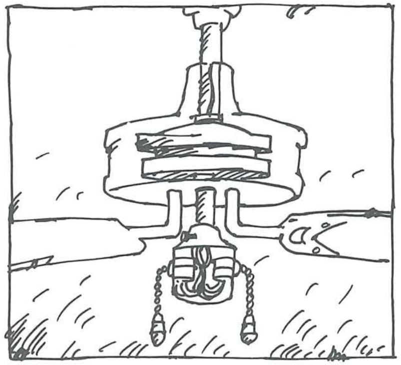
The motor is connected to the rear axle of the car. So when the motor turns, the rear axle will turn — and the car moves.

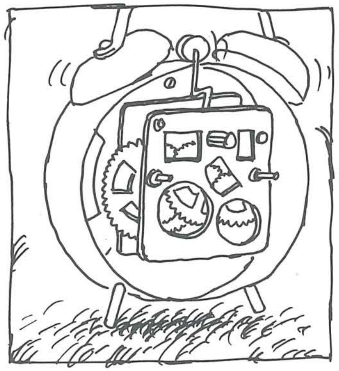
The model uses a "small-to-big" connection. The pulley on the motor is smaller than the pulley on the axle. The axle turns **more slowly** than the motor, but with **greater force**. Trace the motor, pulley and gear connections on your car.

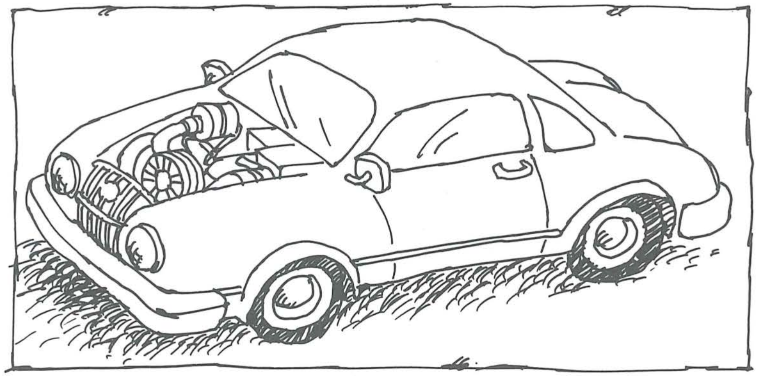
Motors

There are many types of motors. Some motors, like the motors in real cars, get their power by burning **gasoline** or other fuels. Other motors, like the LEGO® motors, use **electricity.**

There are motors in many types of machines. There are motors in cars, in washing machines, in clocks, in merry-go-rounds.







The importance of motors. As the students add motors to their cars, you can discuss the differences between motorized and nonmotorized motion. How would the world be different without motors? How else could you make a car move? (Examples: a horse could pull the car; you could put a sail on top and use the wind.)

You could also talk about the historical importance of motors. What was the world like before motors? How have motors changed the way we live, work, and play?

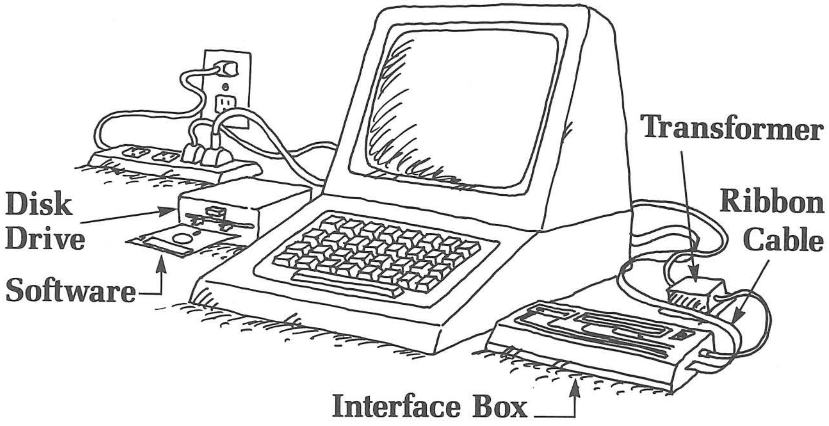
You might ask: How is the computer **unlike** the brain? How are sensors **unlike** our senses?

The LEGO® TC logo System

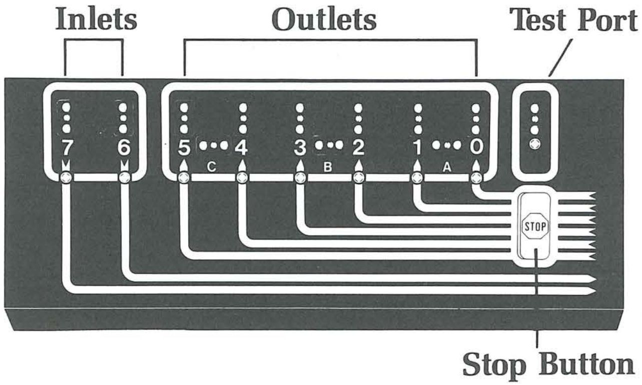
Using the computer, you can **program** the things you build out of LEGO® elements. You can program a car to move forward and back. You can program a traffic light to turn on its red light, its green light, then its yellow light — then start over again!

The LEGO® TC logo system works somewhat like the human body. The computer is like the brain. Your brain sends messages to your muscles, telling them how to move. Similarly, the computer sends messages to LEGO® motors and lights, telling them to turn "on" or "off."

Other LEGO® pieces, known as **sensors**, send messages back to the computer. A touch sensor, for example, is like a finger. If it is being pressed, it sends a message that says "true." If it is not being pressed, it sends a message that says "false." You will learn more about sensors later in this book.



8



The computer communicates with LEGO® pieces through the black **interface box**. The interface box has 12 sets of holes called **ports**. Some of the ports, called outlets, are used to **send** messages (like "on" or "off") to motors and lights. Other ports, called inlets, are used to **receive** messages (like "true" or "false") from sensors.

The red "stop button" lets you stop motors quickly. When you press the button, you cut off power to all outlet ports, so all motors and lights will turn off.

The test port supplies power at all times (even when the "stop button" is pushed). You can use it to test LEGO® machines. Use a LEGO® wire to connect your car to the test port. The car should begin moving. If the car does not run smoothly, look for places where pieces are rubbing against one another.

9

Advantages of computers. This is a good time to discuss the value of computers. The computer gives students control over motors and other devices. They can **program** the car to perform **sequences** of instructions. These instructions can be stored, repeated, and modified. As students begin to write LEGO® TC logo procedures, the power of the computer will become even more obvious.

History and uses of computers. The Bibliography (on pages 151-157 of the *Teacher's Guide*) lists several references. Some questions you might want to discuss: What can we do with computers that we couldn't do before? How are computers changing the way people work? How are they changing entertainment (video games, etc.)? How are they changing education?

These instructions assume that you have already installed the LEGO® interface card. See the *Setup Guide* for details.

LEGO® TC logo software operates very much like the version of Logo called *LogoWriter*®. Students who have experience with *LogoWriter*® will already know the information in this section.

Apple users:

The LEGO® TC logo Operating Disk and Master Disk are identical except that the Master Disk is write-protected. Keep the Master Disk stored safely. Use it only for making back-up copies of the software. To make back-up copies, use any Apple® DOS copy program. Follow the directions in your DOS manual.

The Operating Disk is write-enabled. You can store your own procedures on the Operating Disk.

To start the LEGO® TC logo software, put the Operating disk into your disk drive, with the label facing up. Then turn on the monitor and the computer. After a few seconds, the LEGO® TC logo title screen will appear.

IBM users:

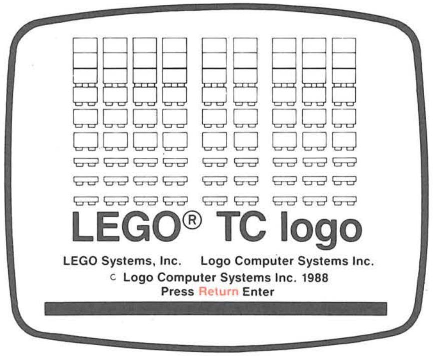
The LEGO® TC logo software includes two Master Disks: a 3½-inch disk and a 5¼-inch disk. Use whichever disk is appropriate for your disk drive.

You will need to make a **user disk** from the Master Disk. There are several ways to do this. The simplest approach is to make a copy of the Master Disk. This user disk will **not** "boot" by itself: you will have to load MS-DOS on your computer first, and then type `tclogo` at the prompt. There is also a way to make a "bootable" user disk. See the *Setup Guide* for details.

Using the Computer

To use the computer, you need the LEGO® TC logo software disk. Different computers start the LEGO® TC logo software in different ways. Look at the *Setup Guide* to find out how to start the LEGO® TC logo software on your computer.

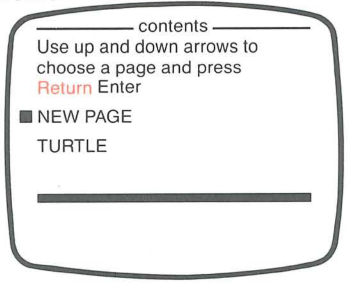
Once the LEGO® TC logo software is loaded, the computer screen will look like this:



LEGO® TC logo

LEGO Systems, Inc. Logo Computer Systems Inc.
c Logo Computer Systems Inc. 1988
Press Return Enter

Press the Return Enter key. (On IBM computers, Enter looks like this: ↵ .) Then wait until you see the Contents menu.



contents

Use up and down arrows to choose a page and press Return Enter

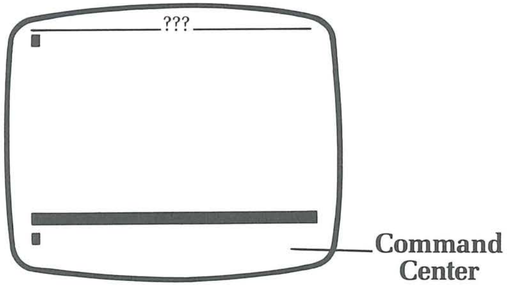
■ NEW PAGE

TURTLE

The LEGO® TC logo disk is organized into **pages**. The Contents menu lists all the pages on the disk.

The blinking light is called the **cursor**. You can move the cursor by pressing the **up arrow** or **down arrow** keys.

Move the cursor next to the words NEW PAGE. Then press the Return Enter key. A new page will appear:



The screen is divided into two parts. The bottom part of the screen is called the **Command Center**. This is where you type instructions to the computer.

Students need not use a motorized car. They could use other motorized constructions, such as a windmill, an electric door, or a fan.

`on` does not indicate a length of time. The motor will stay on until you tell it to turn `off`.

`rd` does not tell the motor to go on. It simply tells the motor to reverse the direction it is set to turn.

Notice that there are ten time units in a second. `onfor 80` will turn on the motor for eight seconds.

Students should experiment with the commands `talkto`, `on`, `off`, `onfor`, `rd` until they feel comfortable with them.

One common mistake: students sometimes substitute letters for numbers. For example, they might use the letter `O` instead of the number `0`. Or they might use the letters `i` or `l` instead of the number `1`. These will not work.

Possible Bugs

There are three major categories of bugs in using the computer:

1. **The computer gives an error message.** The error message will often (but not always) give you a good clue as to what went wrong. Some common mistakes in this category:

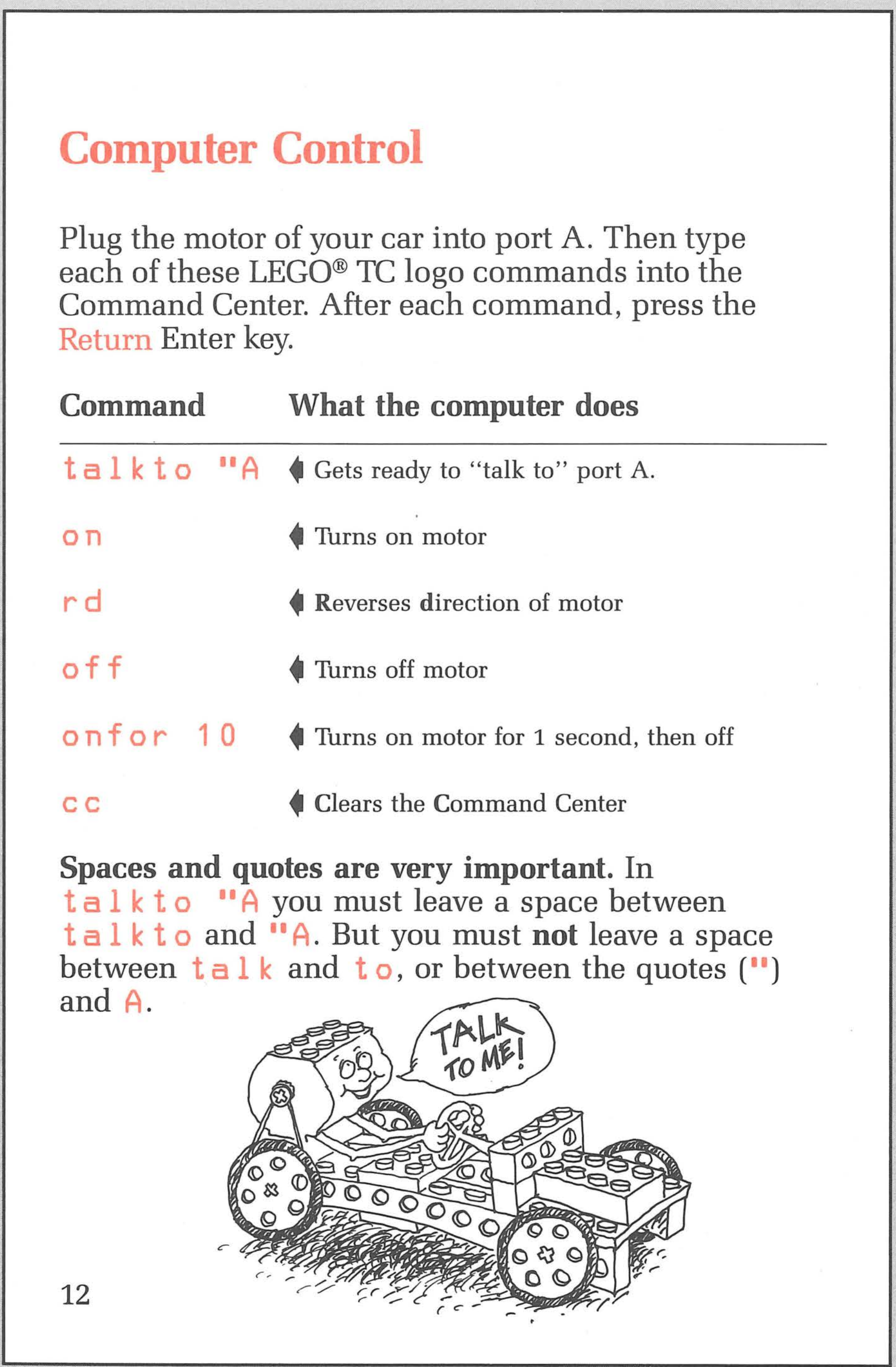
* Spelling a word incorrectly. Sample message: `I don't know how to omfor`.

* Giving an input value to `on`. (Correct: `on` Incorrect: `on 5`) Sample message: `I don't know what to do with 5`.

* Not giving an input value to `onfor`. (Correct: `onfor 5` Incorrect: `onfor`) Sample message: `onfor needs more inputs`.

* `Forgetting the quotes` (") before the name of port A, B, or C. Sample message: `I don't how to A`.

2. **The motor doesn't turn, but there is no error message**. First check whether the motor is plugged into the correct port (or whether it is plugged in at all!). Also, make sure the stop button is not pushed down. Another possibility: maybe nothing

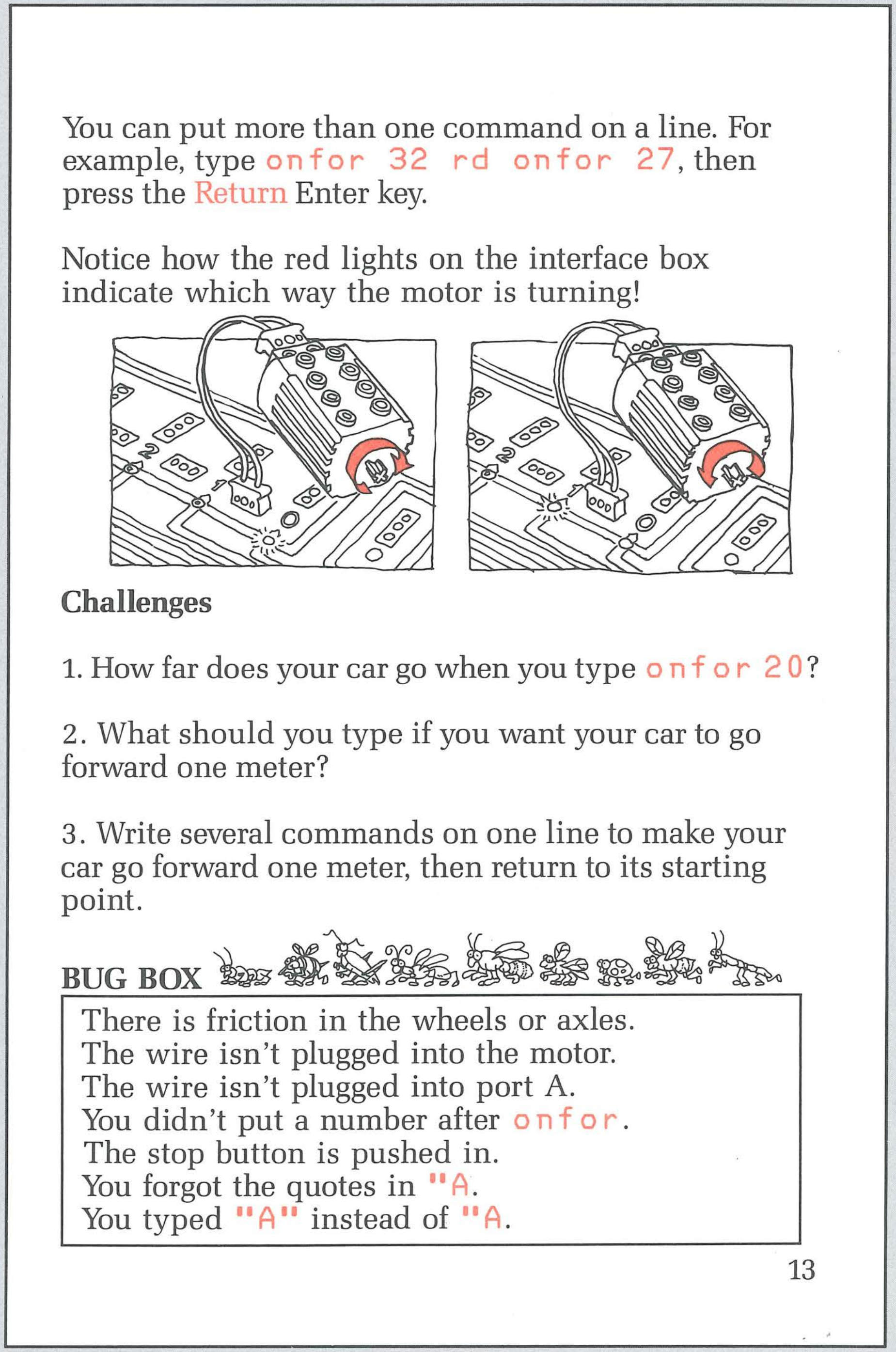


Computer Control

Plug the motor of your car into port A. Then type each of these LEGO® TC logo commands into the Command Center. After each command, press the Return Enter key.

Command	What the computer does
`talkto "A`	Gets ready to "talk to" port A.
`on`	Turns on motor
`rd`	**R**everses **d**irection of motor
`off`	Turns off motor
`onfor 10`	Turns on motor for 1 second, then off
`cc`	Clears the Command Center

Spaces and quotes are very important. In `talkto "A` you must leave a space between `talkto` and `"A`. But you must **not** leave a space between `talk` and `to`, or between the quotes (") and `A`.



You can put more than one command on a line. For example, type `onfor 32 rd onfor 27`, then press the Return Enter key.

Notice how the red lights on the interface box indicate which way the motor is turning!

Challenges

1. How far does your car go when you type `onfor 20`?

2. What should you type if you want your car to go forward one meter?

3. Write several commands on one line to make your car go forward one meter, then return to its starting point.

BUG BOX

There is friction in the wheels or axles.
The wire isn't plugged into the motor.
The wire isn't plugged into port A.
You didn't put a number after `onfor`.
The stop button is pushed in.
You forgot the quotes in `"A`.
You typed `"A"` instead of `"A`.

13

is supposed to happen. For example, if you type `talkto "A`, nothing happens. Similarly, if you type `rd` (while the motor is off), nothing happens.

If the motor is making some noise, but is not moving, there is probably a mechanical problem. A leading suspect is friction.

Apple users: Make sure the interface card is plugged into the correct slot. See the *Setup Guide* for details.

3. **The car doesn't behave as you expected**. There are lots of reasons why this could happen. Computers do what you tell them to do, not necessarily what you want them to do. Look at your commands carefully. Make sure that you understand what each of the commands is supposed to do, and that you are telling the computer exactly what you mean.

Metric measurement. The LEGO® TC logo student books use metric measurements. If your students are more familiar with inches and feet than centimeters and meters, this is a good opportunity to review metric measurement. Page 149 of the *Teacher's Guide* includes information about conversion between English and metric units.

Challenges

1. Use a ruler or other measuring device. (See section on **measurement** on page 31 of the *Teacher's Guide*.)

2. Students can do this by trial and error, or they can calculate the appropriate input for `onfor`. Let's say the car goes 40 centimeters when you type `onfor 20`. Then to make the car go one meter (100 cm.) you can solve this ratio:

$$\frac{\text{20 time units}}{\text{40 cm.}} = \frac{\text{?? time units}}{\text{100 cm.}}$$

So `onfor 50` should make the car go one meter.

3. If `onfor 50` makes the car go one meter, then try:

```
onfor 50 rd onfor 50
```

(The car might not end up exactly at the same place it started, since the motor might be somewhat "stronger" in one direction than the other. You can change the numbers to compensate.)

The quotation marks might cause confusion. Note that there are quotes only **before** the letter, not after.

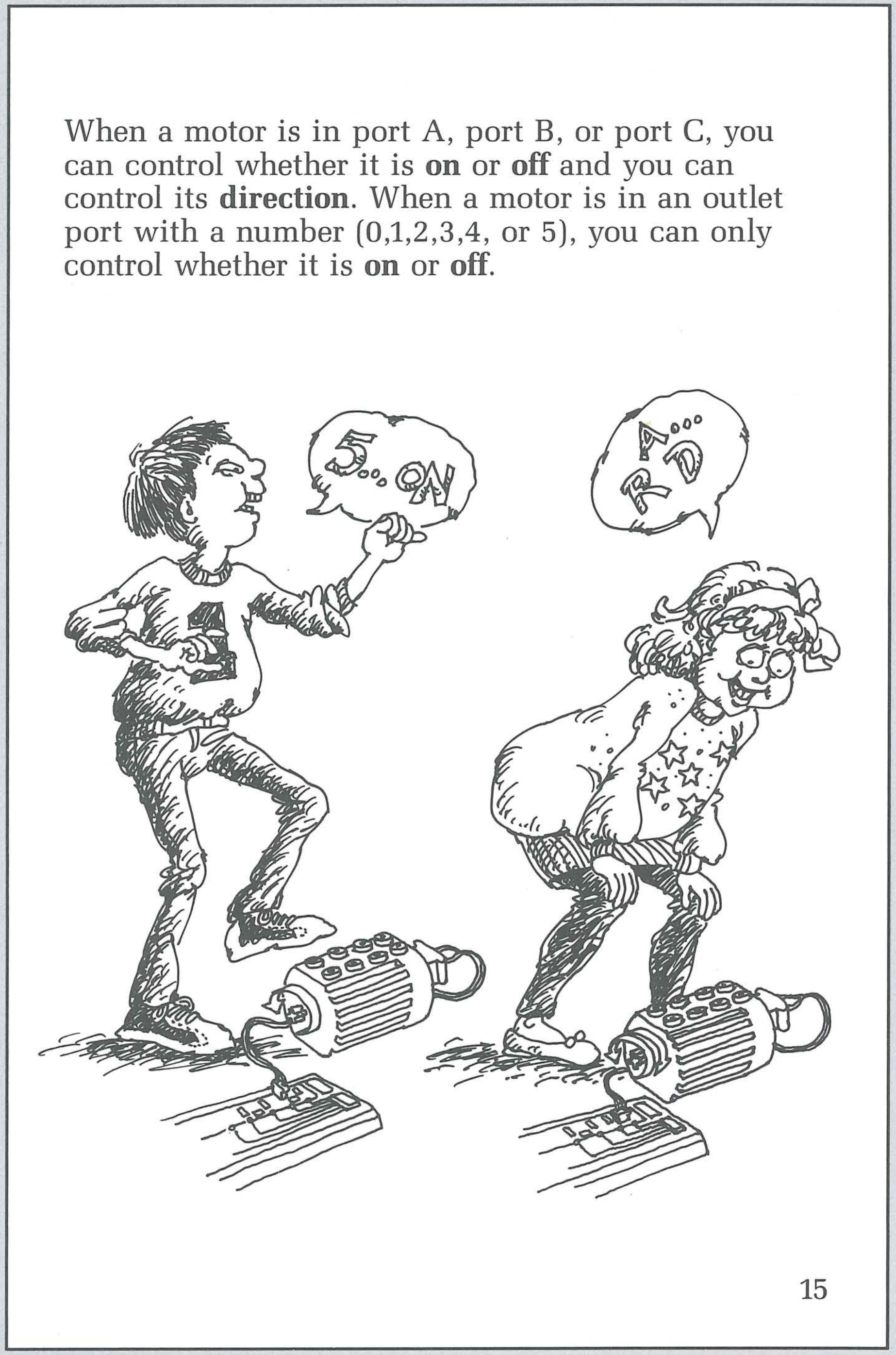
Talking to Other Ports

The computer can give instructions to any of the outlet ports. You must simply tell the computer which port it should "talk to."

Plug one motor into port A and another motor into port C. Then try each of these commands. Remember to press the Return Enter key after each command.

Command	What the computer does
`talkto "C`	Gets ready to "talk to" port C
`on`	Turns on motor in port C
`rd`	Reverses direction of motor in port C
`talkto "A`	Gets ready to "talk to" port A
`onfor 10`	Turns on motor in port A for one second
`talkto "C`	Gets ready to "talk to" port C
`off`	Turns off motor in port C

You can use the abbreviation `tto` instead of `talkto`. So `tto "A` is the same as `talkto "A`



When a motor is in port A, port B, or port C, you can control whether it is **on** or **off** and you can control its **direction**. When a motor is in an outlet port with a number (0,1,2,3,4, or 5), you can only control whether it is **on** or **off**.

15

Port A actually uses information from port 0 and 1. When port 0 is turned on, a motor in port A will turn in one direction. When port 1 is turned on (and port 0 is off), the motor in port A will turn in the other direction. Similarly, port B uses information from port 2 and port 3, and port C uses information from port 4 and port 5.

Because each lettered port uses information from the numbered ports adjacent to it, you cannot use these adjacent ports at the same time. For example, you can not use port A and port 1 at the same time.

Other motor projects. Students can build other simple projects using LEGO® motors. They could work on other types of vehicles (motorcycles, tractors) or on other types of machines (merry-go-rounds, windmills).

Speed. Students can calculate the speed of their LEGO® cars, using the formula: speed = distance/time. (Actually, this formula gives an **average** speed.) When students type `onfor 100`, the car will go for ten seconds. Next, they should measure the distance the car goes. Finally, divide by ten to calculate the car's speed. This approach is better than simply using `onfor 10`, since any errors in measurement are distributed over a larger distance.

Writing procedures. Students with *LogoWriter®* experience can start writing procedures to control the car. For example, a student might write a procedure to send the car forward about one meter, and then return to its starting point. Say that the car takes 5 seconds to go one meter. Then the procedure should look something like this:

```
to meter
talkto "A
onfor 50
rd
onfor 50
end
```

Procedures are introduced and explained in greater detail in the next project ("Traffic Light").

Project 3: Traffic Lights

Key Ideas

- Computer control of lights
- New primitives: `repeat`, `wait`, `alloff` `listento`, `sensor?`, `show`, `waituntil`, `not`, `if`, `ifelse`
- Programming concepts: lists, procedures, recursion
- The purpose of sensors
- Introduction to touch sensors
- Science concepts: Animal senses, light, color

Codes. Everyone understands the code of a traffic light (Red means stop, etc.). But lights can be used for other codes as well. Students could use the traffic light to invent codes of their own. Red can mean one thing, Red/Yellow together can mean another, etc.

In all, you can represent eight different signals using combinations of the three lights. As a challenge, students might try to list all eight possibilities. (The eight possibilities are: Red only, Yellow only, Green only, Red/Yellow, Red/Green, Yellow/Green, Red/Yellow/Green, none).

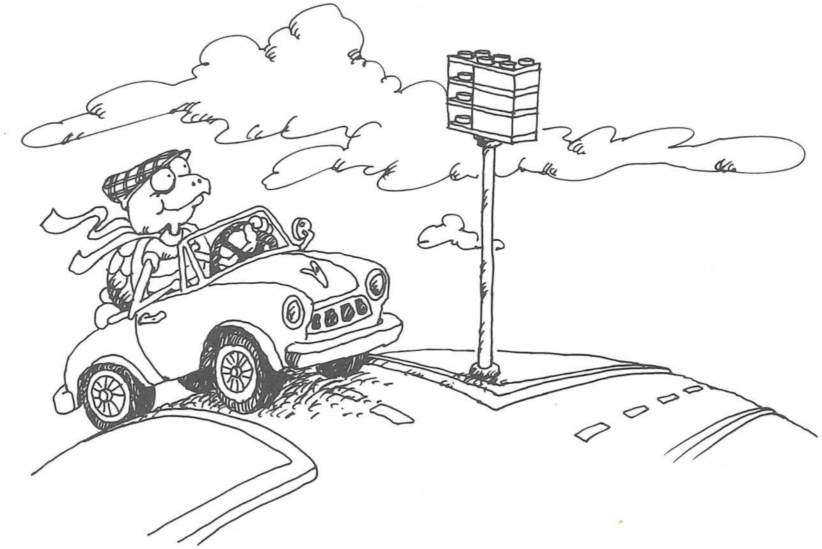
You could also discuss other codes: Morse code, naval codes, Braille.

Light and color. You can use the traffic light as an introduction to the study of light and color. The LEGO® lights produce light waves of many different frequencies. Each frequency corresponds to a different color. We see this combination of frequencies as white light.

When you put a red filter over a light, the filter blocks out all frequencies except those that we see as red. Similarly, a green filter blocks out all frequencies except those that we see as green.

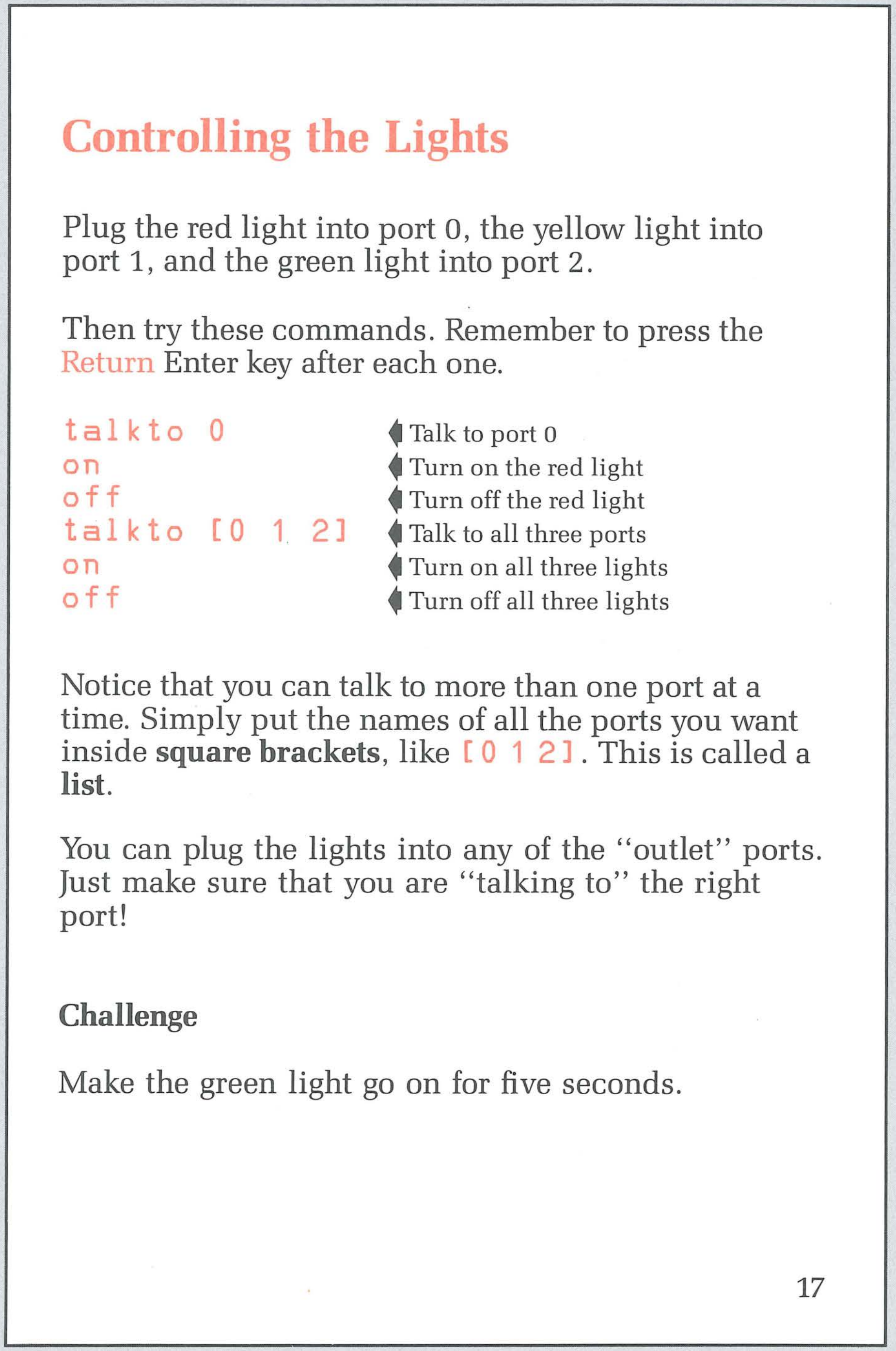
Other lights projects. Students could build other projects using lights. Some possibilities: a house with lights in each room, a car with headlights, a stove with indicator lights, a candelabra with flashing candles.

Project 3: Traffic Lights



Traffic lights coordinate the movement of cars and people. They use colors as codes. Red is the code for Stop. Green is the code for Go.

You can build a LEGO® traffic light and program the computer to control the lights. See Activity Card 2 for instructions on how to build the light.



Controlling the Lights

Plug the red light into port 0, the yellow light into port 1, and the green light into port 2.

Then try these commands. Remember to press the Return Enter key after each one.

`talkto 0`	◀ Talk to port 0
`on`	◀ Turn on the red light
`off`	◀ Turn off the red light
`talkto [0 1 2]`	◀ Talk to all three ports
`on`	◀ Turn on all three lights
`off`	◀ Turn off all three lights

Notice that you can talk to more than one port at a time. Simply put the names of all the ports you want inside **square brackets**, like `[0 1 2]`. This is called a **list**.

You can plug the lights into any of the "outlet" ports. Just make sure that you are "talking to" the right port!

Challenge

Make the green light go on for five seconds.

You don't have to use ports 0, 1, and 2. You could use 0, 2, and 5, or any other combination of three ports.

The **numbered** ports (0-5) control only on/off, while the **lettered** ports (A-C) can also control direction. It is good practice to use numbered ports for lights, and lettered ports for motors.

You must use square brackets for lists. Parentheses won't work.

Lists are an important concept in Logo. By using lists, you can combine many pieces of information into a single unit. Listing makes it easier to manipulate the information at once rather than separately. If students already have some Logo experience, you might want to discuss lists in more detail.

"Real" traffic lights. Students could "time" traffic lights near the school. For each traffic light, they should measure the length of time for the red, for the green, and for the yellow. How do these numbers differ at different intersections? Do busier streets have longer green lights? If not, why not?

Students could also calculate the "green/red ratio" at each light. That is, how does the "green time" compare with the "red time" of the light. You can calculate the ratio by dividing the "green time" by the "red time." When a street with lots of traffic intersects a street with very little traffic, it should have a high green/red ratio.

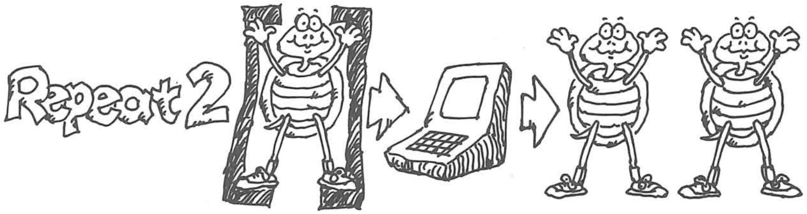
Challenge

```
talkto 2
onfor 50
```

`repeat` takes two inputs. The first input is a **number** which tells the computer how many times to repeat the second input, which is a **list** of commands.

`wait` can be confusing, especially when you are controlling motors. Remember that `wait` tells the computer to pause; it does not tell the motor to pause. If the motor is on, it will stay on. For example, `on wait 30 off` makes the motor stay on for three seconds. It is equivalent to `onfor 30`.

Repeating and Repeating and ...



You can tell the computer to **repeat** a sequence of commands. For example, try this:

```
repeat 2 [talkto 0 onfor 20 talkto 2
onfor 20]
```

The computer will repeat everything that is inside the square brackets (that is, everything in the **list**). The traffic light will go Red-for-two-Green-for-two, then again Red-for-two-Green-for-two.

You can use `repeat` to make a flashing light. Try this:

```
talkto 1
repeat 8 [onfor 10 wait 10]
```

The command `wait 10` tells the computer to wait for one second. So the computer tells the yellow light to go on for one second, then the computer waits for one second (while the light is off). Then the computer repeats the cycle seven more times: on-for-one-off-for-one, on-for-one-off-for-one, . . .

Here's another way to make the light flash. Type `flash 20 10`, and press Return Enter. The light will go on for two seconds, off for one second, over and over. Type `off` to stop the flashing.

The `flash` approach is somewhat different from the `repeat` approach. When you use `flash`, the light will continue to flash "forever" (until you give a new command to the light). Also, you can enter other commands while the light is flashing. For example, try:

```
talkto 1       ◀ Talk to the yellow light
flash 30 5     ◀ Start the light flashing
talkto 0       ◀ Talk to the red light
onfor 40       ◀ Turn on the light for four seconds
talkto 1       ◀ Talk to the yellow light
off            ◀ Stop the flashing
```

Challenges

1. Use `repeat` to make the lights work like a real traffic light: Red, then Green, then Yellow, then Red, then Green, then Yellow.

2. Try changing each of the numbers in
`repeat 8 [onfor 10 wait 10]`
What happens for each change? What happens if you change more than one of the numbers?

3. Make the red light flash quickly (on for a half second, off for a half second), while the green light flashes slowly (on for 3 seconds, off for 3 seconds).

`flash` takes two inputs. The first input tells the light how long to stay on. The second input tells the light how long to stay off.

`flash` works only for numbered ports. If you are talking to port A, B, or C, the computer will ignore the `flash` command.

Challenges

1. Here's one possibility:

```
repeat 2 [talkto 0 onfor 10 talkto 2 onfor 10
talkto 1 onfor 10]
```

You can change the input value of each `onfor` to make the lights stay on more time (or less time). You can change the value of `repeat`'s first input (for example, using four rather than two) to make the traffic light go through its cycle more times.

2. If you change the `8`, you will change the number of times the light flashes. If you change the `10` in `onfor 10`, you will change the length of time the light stays on during each flash. If you change the `10` in `wait 10`, you will change the length of time between flashes.

If you change more than one number, you will get a combination of these effects.

3.
```
talkto 0
flash 5 5
talkto 2
flash 30 30
```

By contrast, the "core" of the English language includes about 60,000 words. Children use a few thousand words in everyday conversation.

Procedures. As students begin writing procedures, you might discuss why procedures are useful. When you write a procedure, you give a name to a set of instructions. Once the set of instructions has a name, you can more easily use it, refer to it, manipulate it, and save it.

Defining New Words

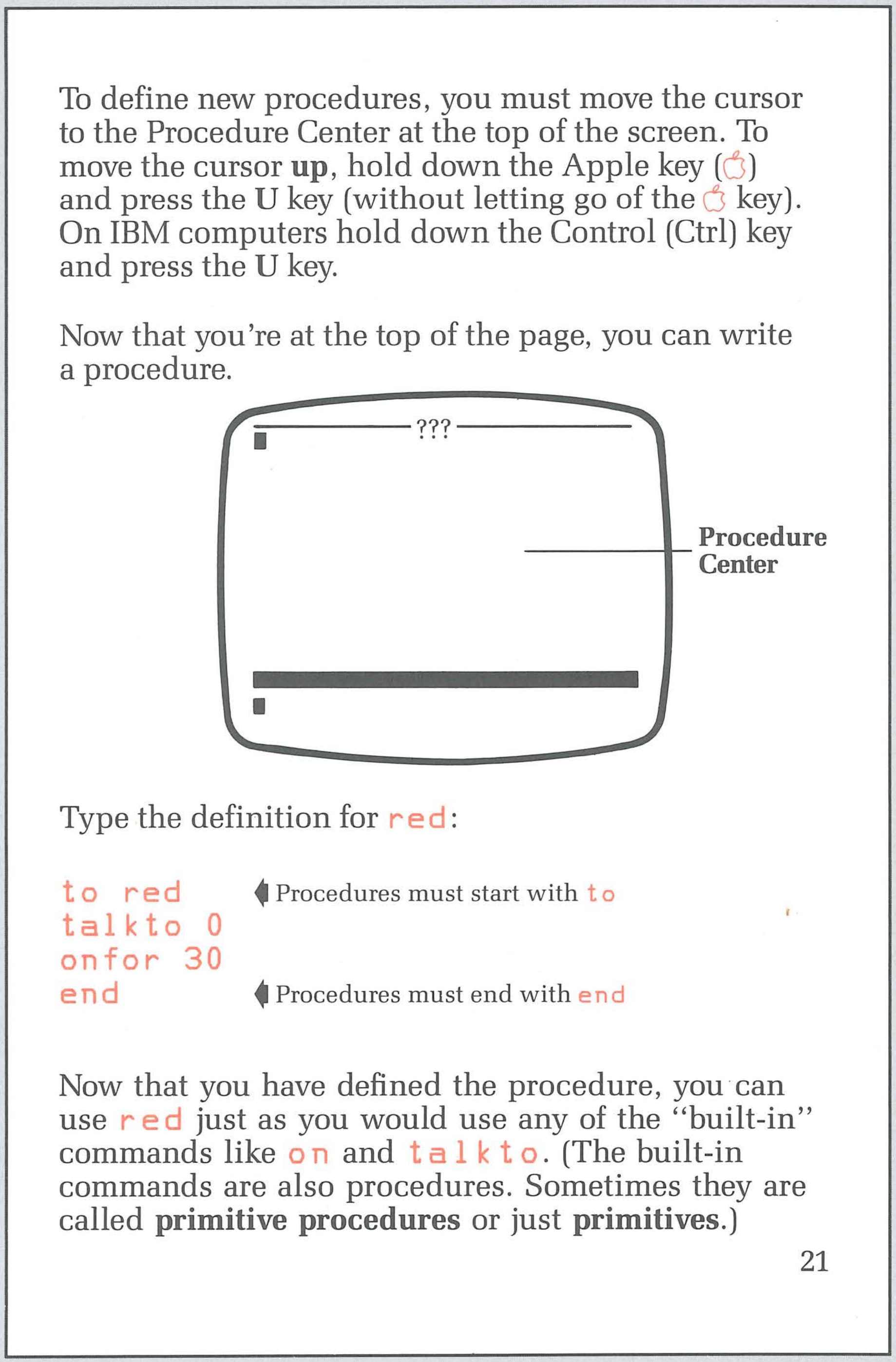
```
TO CAR
TALKTO"A
ON
WAITUNTIL
[SENSOR?]
RD
ONFOR 50
END
```

The computer already understands quite a few commands. It understands `on`, `off`, `talkto`, `repeat` and about 100 others.

That might seem like a lot of commands. But it is often useful to teach the computer **new** commands.

Let's say that you want the red light to go on for three seconds. You could do this by typing `talkto 0 onfor 30`. But wouldn't it be nice if all you had to do was type the word `red`?

To do that, you have to create a **procedure** for `red`. That is, you have to tell the computer exactly what `red` means.



To define new procedures, you must move the cursor to the Procedure Center at the top of the screen. To move the cursor **up**, hold down the Apple key () and press the U key (without letting go of the key). On IBM computers hold down the Control (Ctrl) key and press the U key.

Now that you're at the top of the page, you can write a procedure.

Type the definition for `red`:

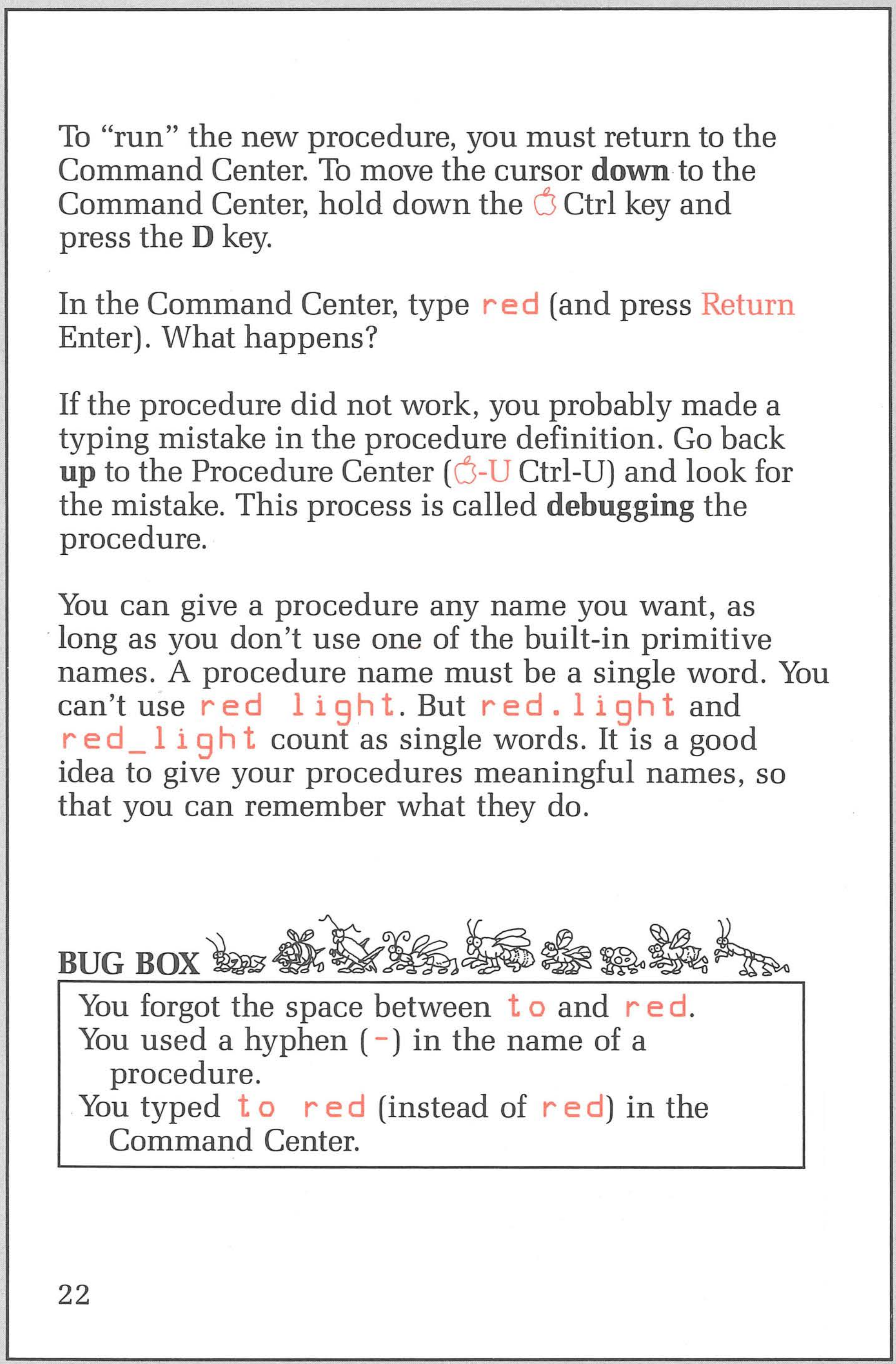
```
to red        ◀ Procedures must start with to
talkto 0
onfor 30
end           ◀ Procedures must end with end
```

Now that you have defined the procedure, you can use `red` just as you would use any of the "built-in" commands like `on` and `talkto`. (The built-in commands are also procedures. Sometimes they are called **primitive procedures** or just **primitives**.)

`to red` must be on a line by itself. `end` must also be on a line by itself. You can format the lines in between any way that you like. For example, you could put `talkto 0` and `onfor 30` on the same line.

The terms "bug" and "debugging" were coined when an actual bug (an insect) got inside one of the earliest computers (in the 1940's) and caused it to malfunction. It is important for students to see "bugs" as a natural part of the programming process. Bugs should not be seen as a sign of failure.

If you use names like `xx`, you might not remember what the procedures are supposed to do. This is a big problem if you try to use or change your procedures a few days (or weeks or months) after you wrote them. It is also important to write procedures that **other people** can understand. That way, you can share procedures and work on joint projects more easily.



To "run" the new procedure, you must return to the Command Center. To move the cursor **down** to the Command Center, hold down the Ctrl key and press the **D** key.

In the Command Center, type `red` (and press Return Enter). What happens?

If the procedure did not work, you probably made a typing mistake in the procedure definition. Go back **up** to the Procedure Center (-U Ctrl-U) and look for the mistake. This process is called **debugging** the procedure.

You can give a procedure any name you want, as long as you don't use one of the built-in primitive names. A procedure name must be a single word. You can't use `red light`. But `red.light` and `red_light` count as single words. It is a good idea to give your procedures meaningful names, so that you can remember what they do.

BUG BOX

You forgot the space between `to` and `red`.
You used a hyphen (`-`) in the name of a procedure.
You typed `to red` (instead of `red`) in the Command Center.

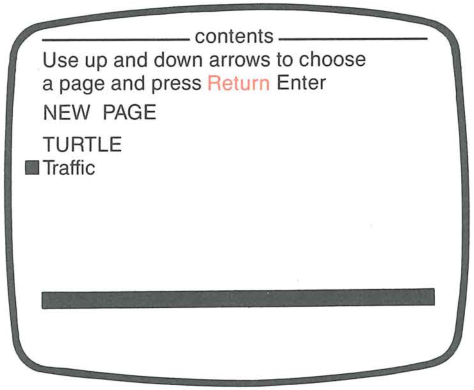
Naming and Saving Pages

Now that you have a procedure on your page, you might want to save the page. That way, you can use the procedure again later.

Before saving the page, you must give the page a name. To name the page **traffic**, type `namepage "traffic` in the Command Center. The new name should appear at the top of the screen! You can use any name you want but don't forget the quotes. (Instead of typing `namepage`, you can simply type `np`.)

To save the page, simply press the Escape (**Esc**) key. The computer returns to the `Contents` menu, and it automatically saves your page.

To return to the `traffic` page, move the cursor to the word `traffic`. (Use the ↓ key.) Then press the Return Enter key.



```
contents
Use up and down arrows to choose
a page and press Return Enter
NEW PAGE
TURTLE
■Traffic
```

The name of a page (like the name of a procedure) must be a single word.

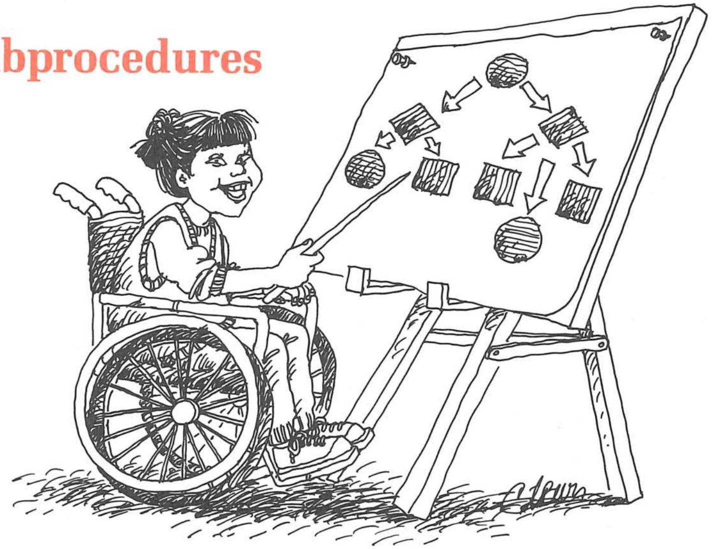
On Apple computers, page names can have at most 12 characters. On IBM and IBM-compatible computers, page names can have at most 8 characters.

Students can write the new procedure definitions right underneath the definition for `red`.

Double traffic light. Build two traffic lights to create an intersection. Write a procedure that controls traffic in both directions. Make sure that both lights are never green at the same time!

Binary counter. Students can use lights to learn about binary arithmetic (base two). When a light is on, it corresponds to the number 1. When the light is off, it corresponds to the number 0. If you plug lights into all six numbered ports (0-5), you can represent every number between 0 and 63.

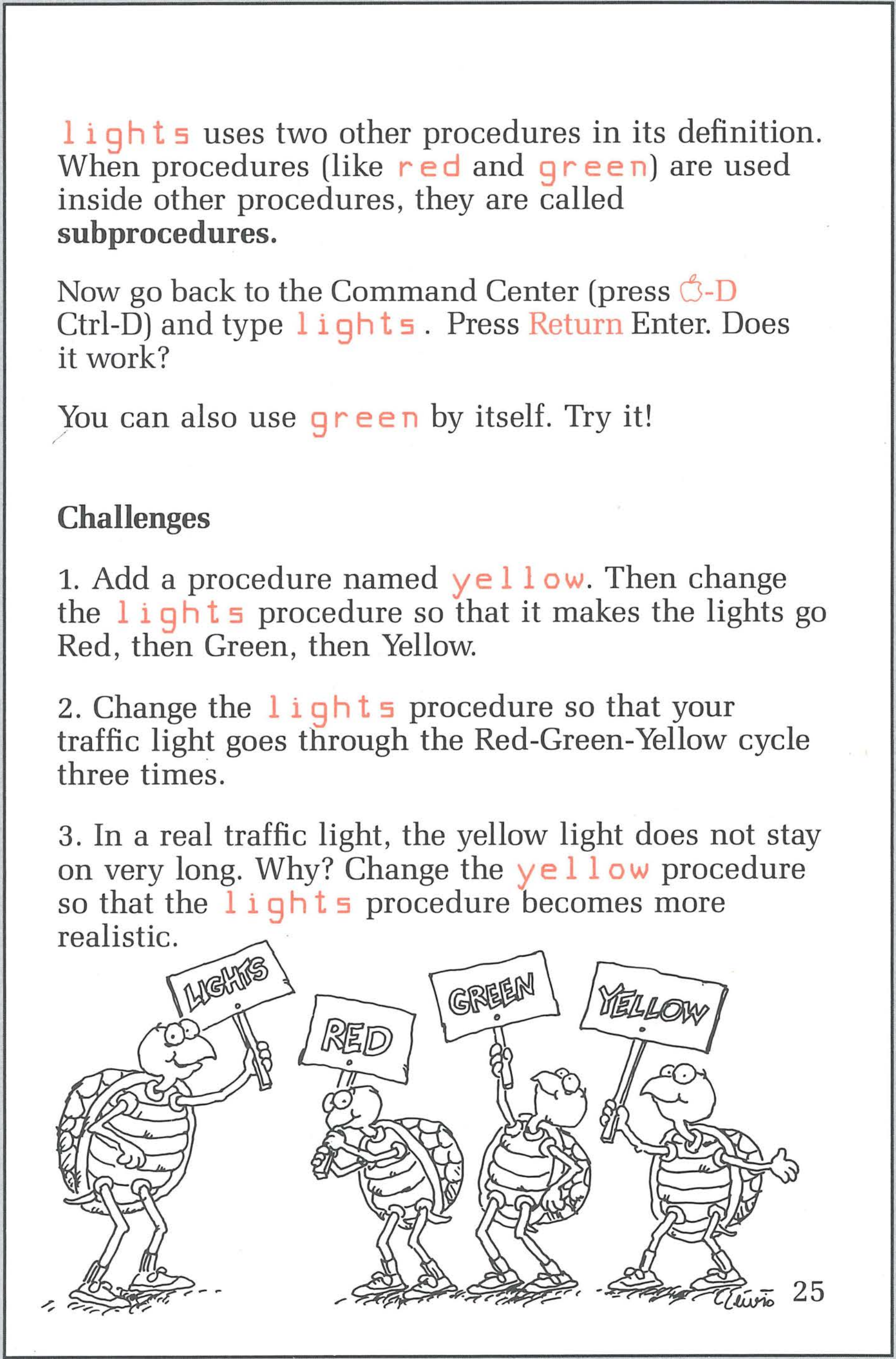
Subprocedures



New procedures can be used just like the "built-in" primitives. For example, you can use the new procedures in the definitions of other new procedures.

Go back to the Procedure Center (press -U Ctrl-U) and add two new procedures:

```
to green
talkto 2
onfor 30
end

to lights
red
green
end
```



`lights` uses two other procedures in its definition. When procedures (like `red` and `green`) are used inside other procedures, they are called **subprocedures.**

Now go back to the Command Center (press -D Ctrl-D) and type `lights`. Press Return Enter. Does it work?

You can also use `green` by itself. Try it!

Challenges

1. Add a procedure named `yellow`. Then change the `lights` procedure so that it makes the lights go Red, then Green, then Yellow.

2. Change the `lights` procedure so that your traffic light goes through the Red-Green-Yellow cycle three times.

3. In a real traffic light, the yellow light does not stay on very long. Why? Change the `yellow` procedure so that the `lights` procedure becomes more realistic.

Challenges

1. Here's one approach:

```
to yellow
talkto 1
onfor 30
end

to lights
red
green
yellow
end
```

2. Students could write:

```
to lights
red
green
yellow
red
green
yellow
red
green
yellow
end
```

But it is easier (and more "elegant") to write:

```
to lights
repeat 3 [red green yellow]
end
```

3. To make the yellow light stay on for just one second:

```
to yellow
talkto 1
onfor 10
end
```

Note that you do not have to make any changes to `lights`!

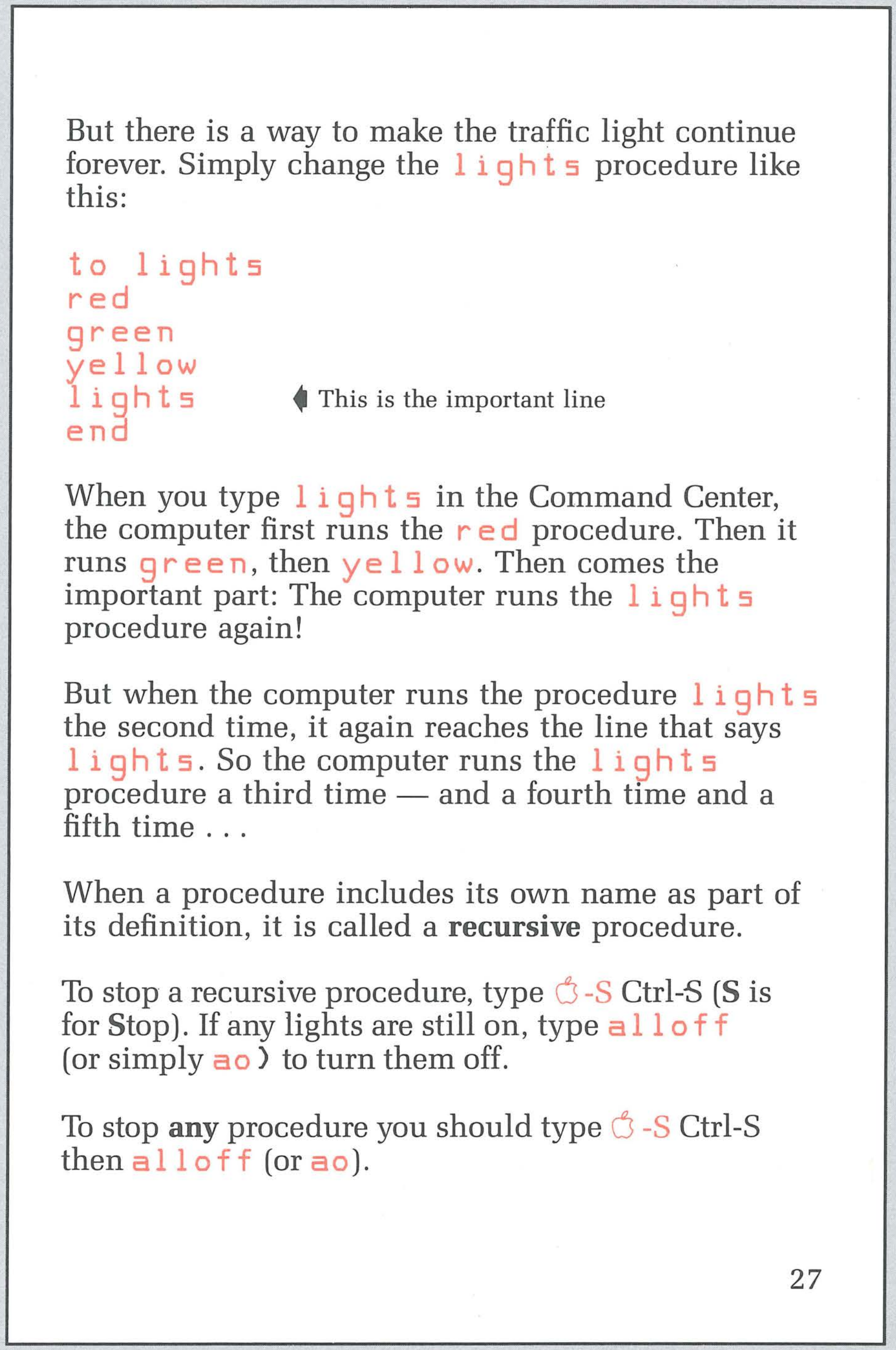
Recursion

You can make the traffic light repeat the Red-Green-Yellow cycle many times. For example,

```
repeat 4 [red green yellow]
```

will make the light repeat the cycle four times (assuming that you've written procedures for `red`, `green`, and `yellow`).

But what if you want the traffic light to repeat the cycle **forever**? No matter how large a number you use with `repeat`, the traffic light will stop eventually.



But there is a way to make the traffic light continue forever. Simply change the `lights` procedure like this:

```
to lights
red
green
yellow
lights        ◀ This is the important line
end
```

When you type `lights` in the Command Center, the computer first runs the `red` procedure. Then it runs `green`, then `yellow`. Then comes the important part: The computer runs the `lights` procedure again!

But when the computer runs the procedure `lights` the second time, it again reaches the line that says `lights`. So the computer runs the `lights` procedure a third time — and a fourth time and a fifth time . . .

When a procedure includes its own name as part of its definition, it is called a **recursive** procedure.

To stop a recursive procedure, type -S Ctrl-S (**S** is for **S**top). If any lights are still on, type `alloff` (or simply `ao`) to turn them off.

To stop **any** procedure you should type -S Ctrl-S then `alloff` (or `ao`).

Here's a more complete explanation: each time the computer runs a procedure, it makes a **copy** of the definition that is in the top part of the page, then it **evaluates** the definition one line at a time.

So to run `lights`, the computer first makes a copy of the definition and begins evaluating it line by line. When the computer reaches the **line** `lights`, it makes a **new copy** of the `lights` procedure and evaluates this new copy line by line. When it gets to the `lights` line in this new copy, it makes yet **another** copy of `lights`, and so on.

Recursion. You might discuss nonprogramming examples of recursion. One example is the story of the magic genie who gave a man three wishes. The man used the first two wishes. Then, for his third wish, he wished for three more wishes!

Another example: A set of two mirrors pointing at one another. The images go deeper and deeper, apparently without end.

-S Ctrl-S simply stops the procedure, not the lights (or motors).

Optosensors are introduced in Project 4 (page 35 of *Getting Started*).

Senses and sensors. You might discuss the similarities and differences between animal senses and machine sensors.

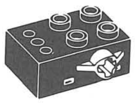
People, of course, have five senses: we see, we hear, we smell, we touch, we taste. Other animals have senses too, but their senses are different than ours. Dogs can smell odors and hear sounds that we could never detect. Honeybees can't see the color red, but they can see other colors that we can't. Bats can detect sound-wave echoes that we can't hear.

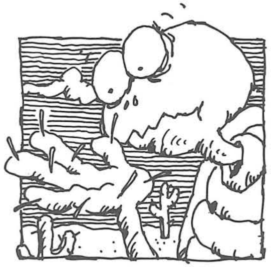
Engineers have built mechanical sensors that can detect almost anything that our senses can detect. There are sensors that detect light, sound, smell, touch. But machines usually can not "make sense" of the signals as well as people can. For example, people can easily tell the difference between a cat and a dog, but even the best computers with "eyes" have difficulty telling the difference. Indeed, by trying to build machines with sensors, scientists have come to realize how very special our own eyes and ears and brains are.

Nevertheless, machines can use sensors to gather all types of useful information. Automatic doors use information from sensors to decide when to open. Smoke detectors use information from sensors to decide when to sound the alarm and turn on the sprinklers.

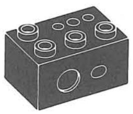
The LEGO® sensors are not nearly as sophisticated as human senses. The touch sensor, for example, can report only `true` or `false`. Our fingers can give us much more information about the things they feel. Imagine if our fingers could only report hot or cold.

Sensors





1. The Touch Sensor



2. The Optosensor

So far, you have used the computer to **send** messages out to LEGO® pieces (motors and lights). But messages can also go in the other direction. The computer can **receive** messages from LEGO® pieces called sensors.

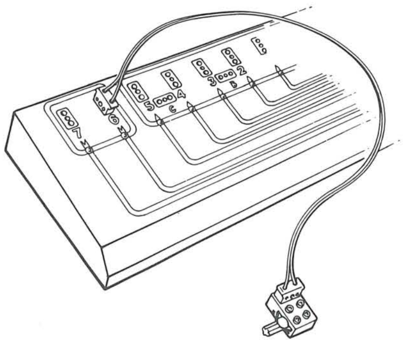
The LEGO® sensors are somewhat like our senses. People have five senses: Sight, smell, hearing, touch, and taste. These senses give us information about what is happening in the world.

LEGO® TC logo has two senses:

The **optosensor** is somewhat like an eye. It tells the computer whether it "sees" light or dark.

The **touch sensor** is somewhat like a finger. It tells the computer whether it "feels" something.

Using a Touch Sensor



You can use a touch sensor to control the traffic light. To start, create a touch sensor by putting a small axle into a touch-sensor brick.

Next, plug the touch sensor into port 6 on the interface box. Then type `listento 6` in the Command Center. This command tells the computer which port to "listen to" to get messages from the sensor.

The sensor will tell the computer whether anything is pressing it. If something is pressing the sensor, the sensor reports `true` to the computer (and the green light under the port goes on). If nothing is pressing the sensor, the sensor reports `false`.

To see what the sensor is reporting to the computer, type `show sensor?` in the Command Center.

Try `show sensor?` several times.

29

"Real" Sensors. Students might look for examples of sensors in the "real" world. Some examples: automatic doors, light meters for cameras, smoke detectors.

`show` tells the computer to print something in the Command Center.

Remember the question mark at the end of `sensor?` (If you type `show sensor`, you will get an error.)

LEGO® TC logo procedures that return either `true` or `false` are sometimes called **predicates**. It is Logo convention to put a question mark at the end of the names of predicates.

Actually, you do not need to retype `show sensor?` over again in the Command Center. Just use the arrow keys to move to that line and press Return Enter. You can do this for any line typed in the Command Center (until you erase the line of course).

Notice that `waituntil` is a single word.

The input to `waituntil` must be in **square brackets**. Parentheses won't work.

The line `waituntil [sensor?]` could be written as `waituntil [sensor? = "true]`. Both do the same thing. (Notice the quotes before `true`.)

Push-Button Control

You are now ready to use the touch sensor to control the traffic light.

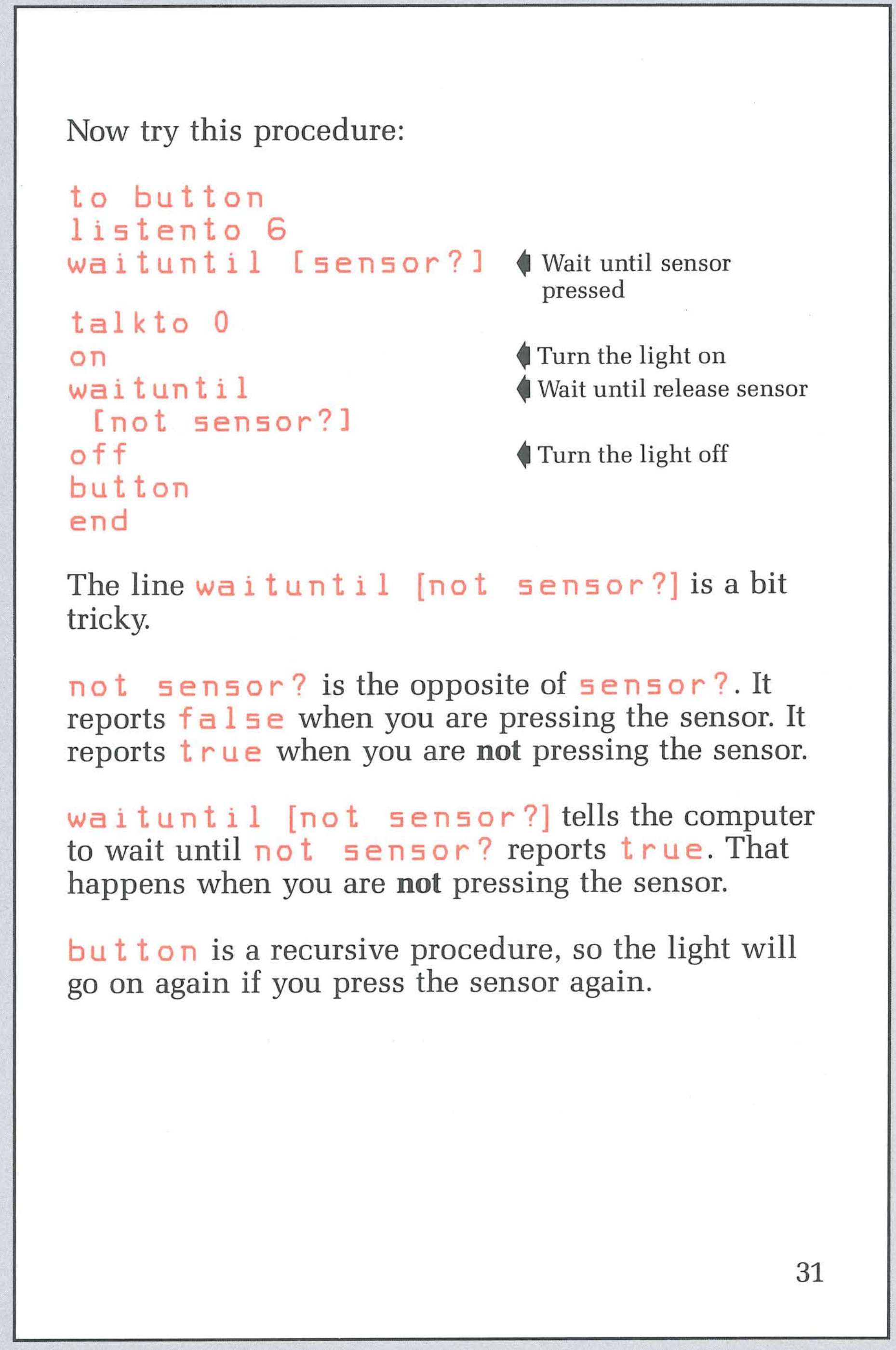
Make sure that the traffic light is connected as it was before: Red in port 0, Yellow in port 1, Green in port 2. Also, make sure the touch sensor is plugged into port 6. Then write this procedure:

```
to switch
listento 6
waituntil [sensor?]
talkto 0
on
end
```

`waituntil` is a LEGO® TC logo primitive that tells the computer to "wait until" the expression in the square brackets reports `true`.

In this case, the computer waits until `sensor?` reports `true` (that is, it waits until you press the sensor). Then the computer turns `on` the red light.

Give it a try. Type `switch`. When you press the sensor, the red light should turn on. (If you change `talkto 0` to `talkto 2` in the `switch` procedure, the green light will go on when you press the sensor.)



Now try this procedure:

```
to button
listento 6
waituntil [sensor?]   ◀ Wait until sensor
                        pressed
talkto 0
on                    ◀ Turn the light on
waituntil             ◀ Wait until release sensor
 [not sensor?]
off                   ◀ Turn the light off
button
end
```

The line `waituntil [not sensor?]` is a bit tricky.

`not sensor?` is the opposite of `sensor?`. It reports `false` when you are pressing the sensor. It reports `true` when you are **not** pressing the sensor.

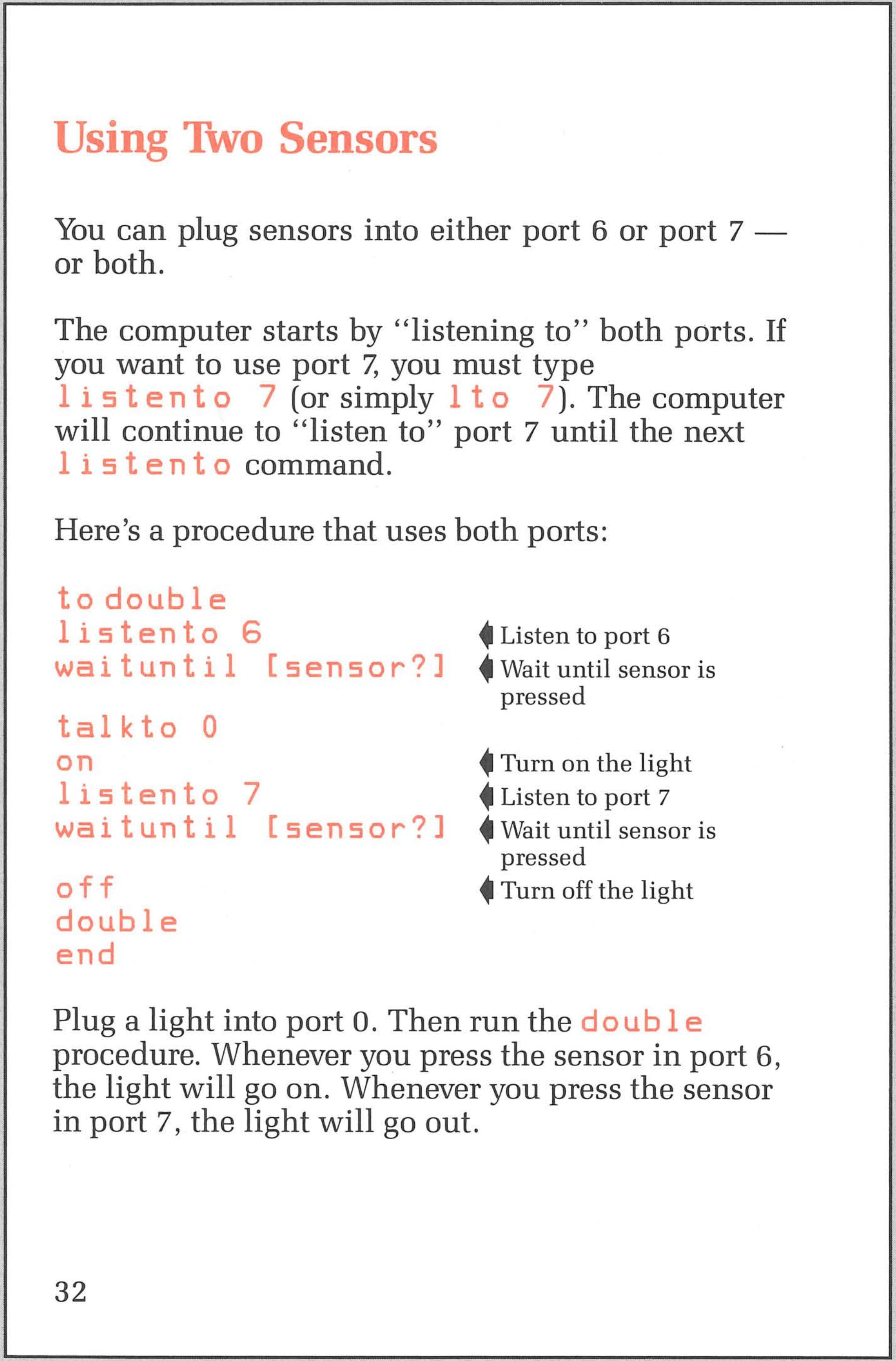
`waituntil [not sensor?]` tells the computer to wait until `not sensor?` reports `true`. That happens when you are **not** pressing the sensor.

`button` is a recursive procedure, so the light will go on again if you press the sensor again.

The line `waituntil [not sensor?]` could be written as `waituntil [sensor? = "false]`. Both do the same thing. (Note the quotes before `false`.)

Other touch-sensor projects. Students can use the touch sensor in other projects. For example, they might use the touch sensor to control the movement of a LEGO® car. Or they could use a touch sensor to control an automatic door. In fact, they could use a touch sensor to control just about any machine that they build.

You use `talkto` to designate an "outlet" port (0-5, A-C) for motors or lights.
You use `listento` to designate an "inlet" port (6,7) for sensors.



Using Two Sensors

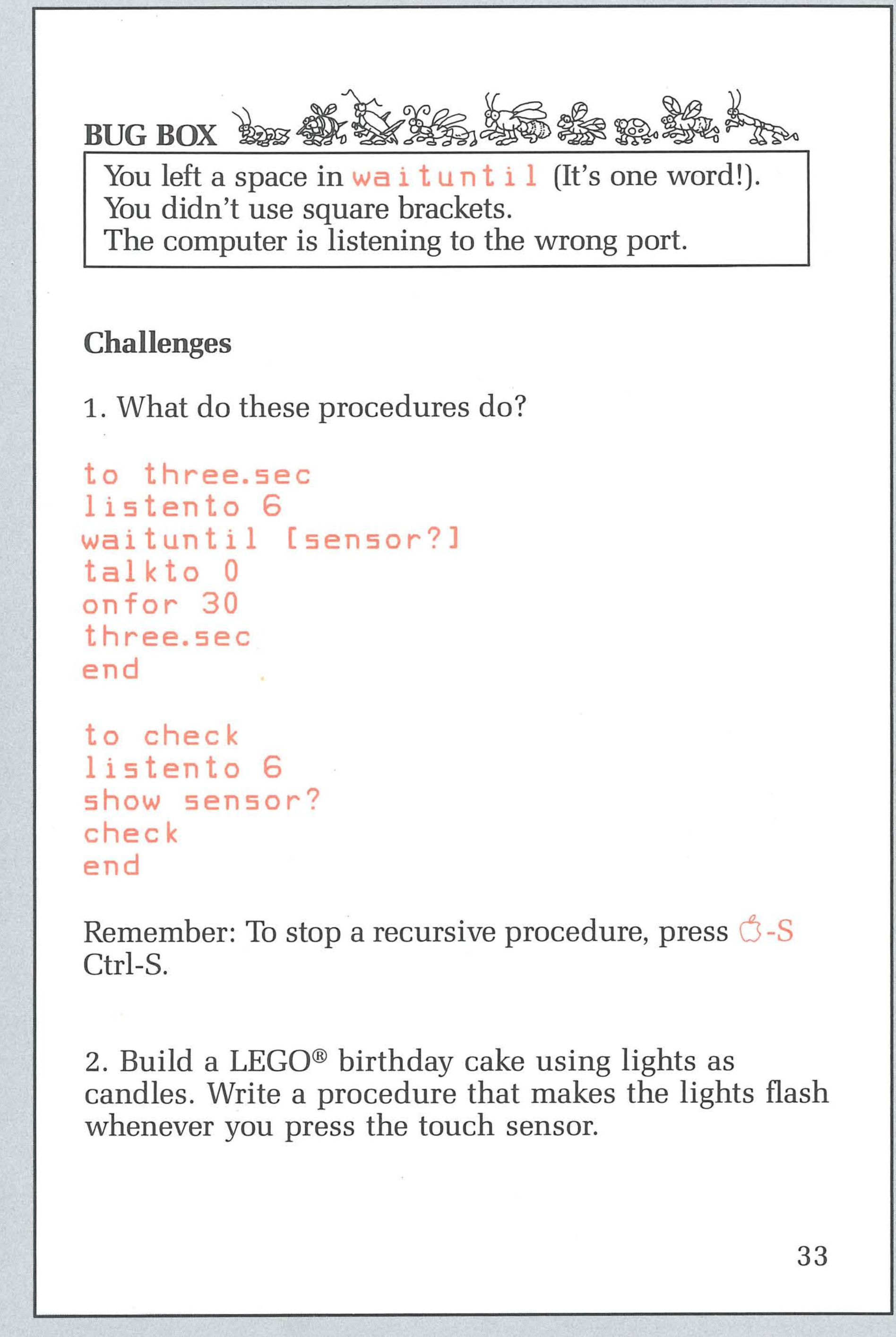
You can plug sensors into either port 6 or port 7 — or both.

The computer starts by "listening to" both ports. If you want to use port 7, you must type `listento 7` (or simply `lto 7`). The computer will continue to "listen to" port 7 until the next `listento` command.

Here's a procedure that uses both ports:

```
to double
listento 6                  ◀ Listen to port 6
waituntil [sensor?]         ◀ Wait until sensor is
                              pressed
talkto 0
on                          ◀ Turn on the light
listento 7                  ◀ Listen to port 7
waituntil [sensor?]         ◀ Wait until sensor is
                              pressed
off                         ◀ Turn off the light
double
end
```

Plug a light into port 0. Then run the `double` procedure. Whenever you press the sensor in port 6, the light will go on. Whenever you press the sensor in port 7, the light will go out.



BUG BOX

You left a space in `waituntil` (It's one word!).
You didn't use square brackets.
The computer is listening to the wrong port.

Challenges

1. What do these procedures do?

```
to three.sec
listento 6
waituntil [sensor?]
talkto 0
onfor 30
three.sec
end
```

```
to check
listento 6
show sensor?
check
end
```

Remember: To stop a recursive procedure, press -S Ctrl-S.

2. Build a LEGO® birthday cake using lights as candles. Write a procedure that makes the lights flash whenever you press the touch sensor.

Challenges

1. When you run the `three.sec` procedure, the light will turn on for three seconds every time you press the sensor. Note that you do not have to hold the sensor down. Just press it and release.

The `check` procedure keeps printing the value of `sensor?` (`true` or `false`) over and over. This type of procedure is useful for testing a sensor. To stop the procedure, press -S Ctrl-S.

2. Here's one possibility:

```
to birthday
listento 6
waituntil [sensor?]             ◀ Wait until sensor is pressed
talkto "A
repeat 6 [onfor 3 wait 1]       ◀ Flash on and off six times
birthday
end
```

Make sure the sensor and motor are plugged into the right ports before you run `birthday`.

Other Approaches

There is usually more than one way to write a procedure. For example, `button` (page 31) can be written (at least) two other ways:

```
to button2
listento 6
talkto 0
if sensor? [on]
if not sensor? [off]
button2
end

to button3
listento 6
talkto 0
ifelse sensor? [on] [off]
button3
end
```

These procedures use the primitives `if` and `ifelse`. `ifelse` covers two cases in one line. If `sensor?` reports `true`, the computer runs the first list (`on`). If `sensor?` reports `false`, the computer runs the second list (`off`).

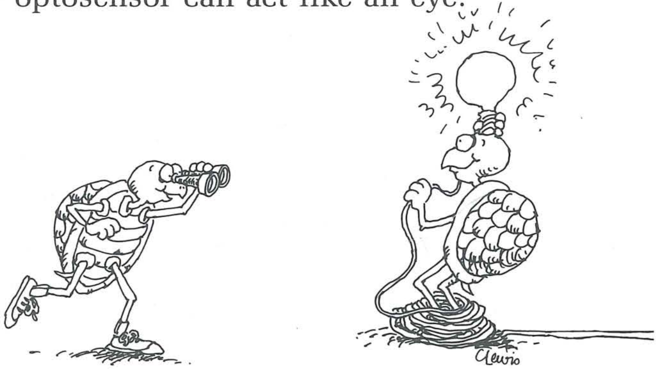
All three `button` procedures do the same thing: They turn the light on when you push the sensor, and turn the light off when you release the sensor. Which approach is best? Sometimes `waituntil` seems easier, sometimes `if` or `ifelse` seems easier. You should use whichever one seems easiest to you.

`if sensor? [on]` checks to see if `sensor?` reports `true`. If so, the computer turns on the light. If not, the computer goes directly to the next line. Unlike `waituntil`, it does **not** wait until the sensor reports `true`.

These books tend to use `waituntil` rather than `if` or `ifelse`. We think that procedures using `waituntil` are a bit easier to understand.

Project 4: Electric Eye

Just as the touch sensor can act like a finger, the LEGO® optosensor can act like an eye.



Of course, the optosensor's "vision" is not nearly as good as your vision. Your eyes can see shapes and colors and shades of brightness. The optosensor can only tell the difference between "dark" and "light."

Still, the LEGO® optosensor can be very useful. Imagine an optosensor on one side of a doorway and a LEGO® light on the other side. When nothing is in the doorway, the optosensor sees "light." But when someone moves through the doorway (and blocks the light), the optosensor sees "dark."

You can write many programs using the optosensor. Your program could sound an alarm if anyone walks through the door. Or it could count how many people walk through the door. Or it could measure how long it takes someone to walk through the door.

Project 4: Electric Eye

Key Ideas

- Introduction to optosensors
- Introduction to the LEGO® timer
- New primitives: `timer`, `resett`

Vision. The optosensor is very different from our eyes. This is a good opportunity to discuss how our eyes work.

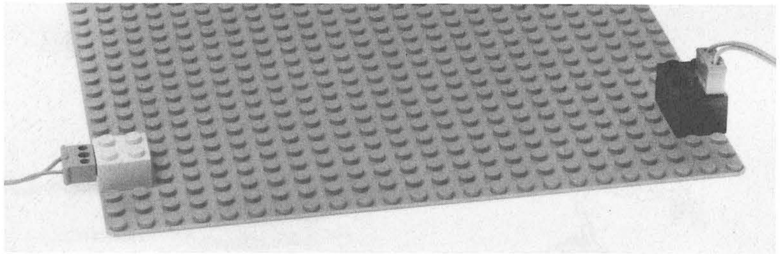
See Chapter 2 of the *Reference Guide* for a more complete explanation of how the optosensor works.

Optosensors might not behave reliably if there is too much light in the room. The sensor might confuse the natural light with the light from the LEGO® light.

The optosensor might not work reliably if it is more than 30 centimeters away from the LEGO light.

Using the Optosensor

Like the touch sensor, the optosensor always reports either `true` or `false` to the computer. The optosensor actually acts like a "dark" sensor. When it sees "dark", it reports `true`. When it sees light, it reports `false`.



Set up a LEGO® light across from an optosensor on a flat LEGO building plate, as shown above. The light and sensor should be about 20 studs apart. Plug the light into port 1 and the optosensor into port 6. Now move the cursor to the top part of the page (-U Ctrl-U) and type in this procedure:

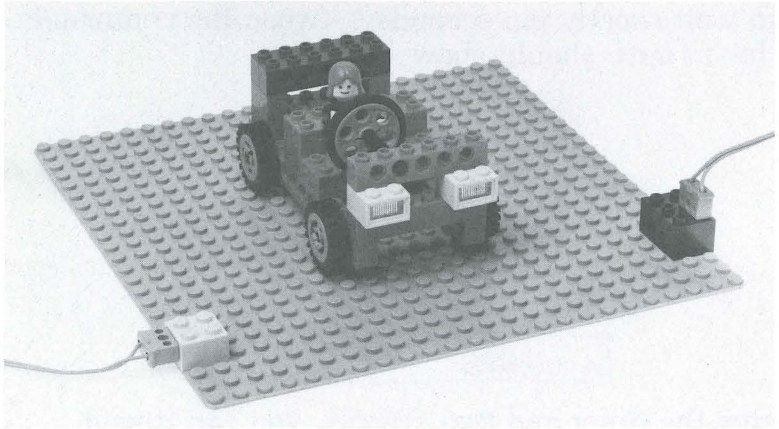
```
to burglar
listento 6
waituntil [sensor?]      ◀ Wait until sensor sees
                           "dark"
show "Burglar!!          ◀ Print out a message
end
```

Return to the Command Center. Type `talkto 1 on` to turn on the light. Then try `burglar`. When you block the light, the computer should print the message.

If you put `talkto 1 on` inside the `burglar` procedure the procedure might not work, since the computer might check `sensor?` before the light has had time to fully "light up."

BUG BOX

You forgot to turn on the light.
The sensor is in the wrong port.
The LEGO® light is too far from the sensor.
There is too much light in the room.
The "burglar" isn't blocking the light completely.



Challenge

Write a procedure that turns on a car motor, but makes the motor stop if the car passes between the light and the sensor.

More electric eyes. Students can develop many other uses for electric eyes. For example, they can build an automatic door that opens whenever anyone walks toward it. They can also think about "real world" uses for electric eyes (Example: monitoring the flow of traffic on a street).

Challenge

```
to car
talkto "B                  ◀ Talk to the motor
on
listento 6
waituntil [sensor?]        ◀ Wait until light is blocked
off
end
```

Turn on the light (`talkto 1 on`), then run `car`.

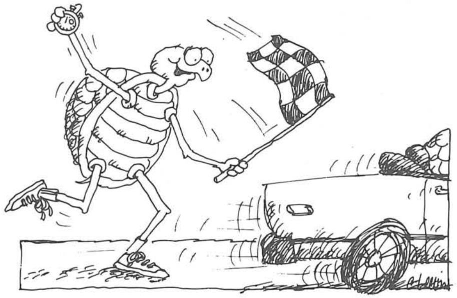
Using the Timer

You can use sensors to start or stop the LEGO®TC logo timer.

To reset the timer to zero, type `resett` in the Command Center (for `reset t`imer). Wait a few seconds then type `show timer`.

Notice that the timer counts in tenths of seconds. So in one second, the timer counts to 10. In two seconds, it counts to 20.

Try `resett` and `show timer` again. See if you can wait exactly ten seconds between the commands. (The `timer` should show `100`.)



Using the timer and two sensors, you can time a Soap Box Derby race. Use the soap box car you built earlier or build a new one.

Plug the touch sensor into port 7. This sensor can act as the "starting gun." The timer should start when you press the sensor.

After ten seconds, `show timer` should print `100`.

The LEGO® TC logo system actually includes **four** timers. The primitives `timer` and `resett` refer to one of the timers. To use the other timers, you must use the primitives `timer#` and `resett#`. See the *Reference Guide* for more details.

Clocks and time. People have been fascinated with clocks and time for thousands of years. The earliest clocks depended on nature (for example, sundials depended on the position of the sun in the sky). Many modern clocks (including the one on the LEGO® interface card) keep very accurate time based on the oscillations of a crystal.

You might want to discuss what life was like before the invention of accurate clocks. How would life be different today if we didn't have clocks?

Building clocks. Students can build many other types of clocks in the classroom. Some examples: a swing pendulum; water dripping out of a hole in the bottom of a cup. Students should discuss the advantages and disadvantages of each approach.

Getting a feel for time. Many LEGO® TC logo commands are based on time. `onfor` is a good example. The input to `onfor` is in tenths of seconds. This leads to questions such as: What should you type to make the motor stay on for an hour? If you typed `onfor 9999` at the end of the school day (and left the computer on), would the motor still be on the next morning? You could actually try this!

You can use the electric eye as the finish line. (Set it up as described on page 36.) The timer should stop when the car passes in front of the electric eye.

Your procedure for the Soap Box timer should look something like this:

```
to time.race
talkto 1
on                        ◀ Turn on the light
listento 7                ◀ Listen to the touch sensor
waituntil [sensor?]       ◀ Wait until sensor is pressed
resett                    ◀ Start the timer
listento 6                ◀ Listen to the optosensor
waituntil [sensor?]       ◀ Wait until sensor "sees"
                            car
show timer                ◀ Print the time
off                       ◀ Turn off the light
end
```

Make sure to push the touch sensor at the same time you let the Soap Box car go!

Challenge

Change the race so that you use electric eyes at both the starting line and the finish line. With an electric eye at the starting line, you do not have to worry about pushing the touch sensor at the beginning of the race. Do you have to change the procedure `time.race`?

Starting gate and finish line. Activity Card 4 shows how you can use LEGO® pieces to build a more sophisticated starting gate and finish line. The starting gate is controlled by a motor. To raise the starting gate, you simply turn on the motor for a second or two. The finish line uses an optosensor and a counting wheel. The LEGO® counting wheel is discussed on page 47 of *Getting Started*.

Remember that the time will be in **tenths of seconds**. If you want the computer to print the time in **seconds** change the line `show timer` to `show timer / 10`.

Challenge

Note: You need two optosensors to do this challenge. If you have only one optosensor, you should skip this challenge.

You don't have to change the procedure. Simply add a light and an optosensor at the starting line. Make sure to turn on both LEGO® lights. If the lights are plugged into port 0 and port 1, you should type `talkto [0 1] on` before running `time.race`.

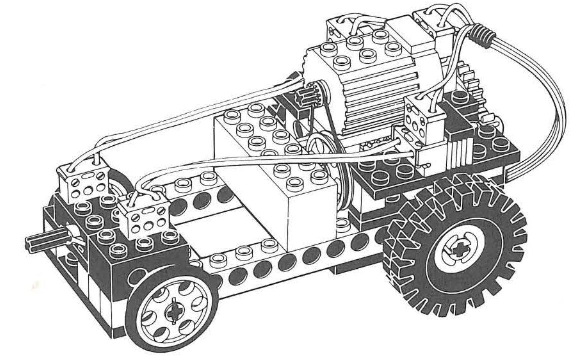
Project 5: Smart Cars

Key Ideas

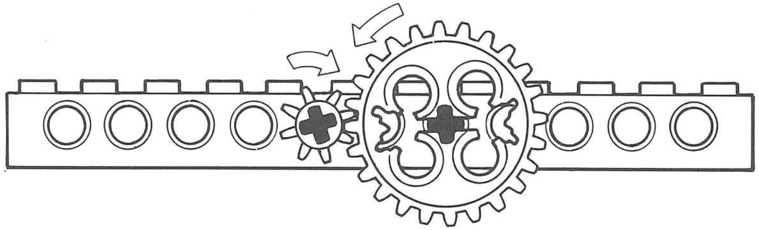
- Introduction to gear trains
- The speed/strength tradeoff
- Mechanical advantage
- Sensors on machines
- Introduction to the counting wheel
- New primitives: `resetc`, `counter`
- New programming concept: procedures with inputs

Gears in the "real world." Students can look for examples of gears in everyday objects. One good activity is to take apart an analog clock. (Try to find one that doesn't work anymore!) By tracing through the gear combinations, you can see why the different hands turn at different rates. You might also look at the electric meter at the school (or have students look at the meters at their homes). In many electric meters, you can see the gears turning. (In fact, adjacent dials on the meter usually turn in opposite directions, since adjacent "meshing" gears turn in opposite directions.)

Project 5: "Smart" Cars



The car you built from Activity Card 3 has four gears and two pulleys. What is the advantage of using so many gears? One reason for using gears is to make a machine **faster** or **slower**.

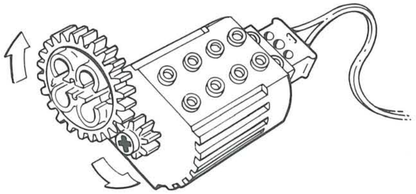


A combination of gears is called a **gear train**. To build the one shown above, place one axle through an 8-tooth gear and another axle through a 24-tooth gear. Then put the axles through the holes on a 12-stud beam. Make sure the teeth of the gears are meshed together properly. Spin one of the axles. The small gear will turn **faster**. Turn the big gear around once and count how many times the small gear turns.

Although the big gear turns more slowly, it is **stronger**. It would be better for driving a car up a hill, or lifting a heavy object.

There is a **tradeoff** between speed and strength. When a small gear and a big gear are meshed together, the small gear is faster and weaker. The big gear is slower and stronger.

You can feel this tradeoff when you ride a ten-speed bicycle. In high gear, you can go fast, but pedaling is difficult. In low gear, you go more slowly, but pedaling is easier.



By using LEGO® gears, you can change the speed and strength of a LEGO® car. LEGO® motors are very fast but very weak. If you connect a motor directly to the wheel of a car, it might not even have enough strength to make the car move.

You can make the car stronger (and slower) by using a **small-to-big** combination of gears, as shown above.

If the big gear has three times as many teeth as the small gear, then the wheel will be three times stronger than the motor. But the motor will turn three times faster.

41

Mechanical advantage. Building gear trains is one way to create a "mechanical advantage." If a machine has a mechanical advantage greater than one, then it can produce a larger force than it starts with.

For example, if you split a seesaw so that two-thirds of the plank is on one side of the fulcrum (the balance point), then you can lift 50 kilograms (about 110 pounds) by supplying only 25 kilograms (about 55 pounds) of force. So the seesaw has a mechanical advantage of two. The tradeoff is distance: the 50 kilogram weight will move only half as far as you move.

Similarly, if you use an 8-tooth gear to drive a 16-tooth gear, the gear train has a mechanical advantage of two. So the "output" gear (the 16-tooth gear) puts out a force twice as large as you put in — but it moves at only half the speed. (If you use a 16-tooth gear to drive an 8-tooth gear, the gear train has a mechanical advantage of 0.5. You get out a force half as large as you put in — but the output gear moves at twice the speed.)

If you use a **big-to-small** combination, the wheels of the car will turn faster than the motor, but with less strength. When you hold the car in your hand, the wheels will turn very quickly. But when you put the car down, the wheels probably won't have enough strength to make the car move.

You might explain this in terms of **tooth steps**. Consider a 24-tooth gear "meshed" with an 8-tooth gear. When the 24-tooth gear makes one complete revolution, it takes 24 tooth steps. The 8-tooth gear must also take 24 tooth steps. But that means that 8-tooth gear makes 3 revolutions ($24/8 = 3$). So the 8-tooth gear turns three times as fast as the 24-tooth gear.

Imagine that you turn the fourth axle (on the far right) one revolution. The third axle turns twice, since its 8-tooth gear has half as many teeth as the 16-tooth gear on the fourth axle.

But every time the third axle turns once, the second axle turns three times (since the gear on the third axle has three times as many teeth as the gear on the second axle). So when the third axle turns twice, the second axle turns six times.

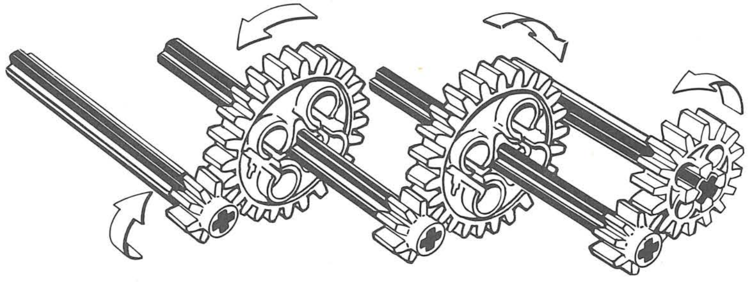
Similarly, when the second axle turns once, the first axle turns three times (24/8 = 3). So when the second axle turns six times, the first axle turns 18 times.

Fractions and ratios. Gears provide a tangible and meaningful environment in which students can explore fractions and ratios. Putting gears together into a gear train provides a "real-world" example of multiplying fractions. For example, this gear train has a mechanical advantage of:

$$\frac{24}{8}\times\frac{24}{8}\times\frac{16}{8}=18$$

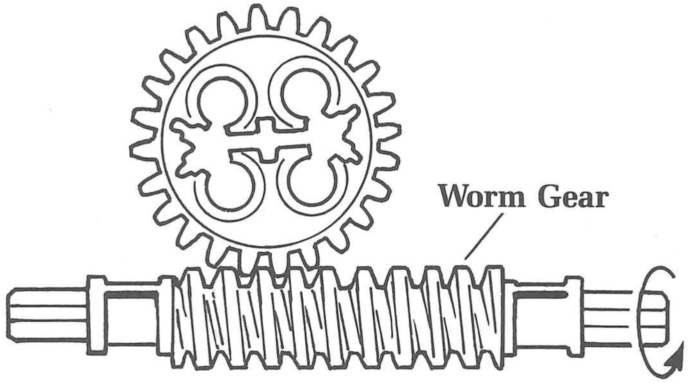
Every time the worm gear turns one revolution, the gear it is "meshing" with advances by only one tooth.

To make an even bigger change you can build a gear train with more than two gears.



Every time gears "mesh" together, the speed and strength changes. When it goes small to big, it becomes slower and stronger. When it goes big to small, it becomes faster and weaker.

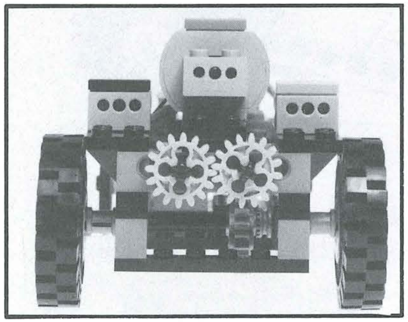
Worm gears are particularly useful in gear trains. When a worm gear is meshed with another gear, the worm gear is always faster and weaker. So the worm gear acts like a very "small" gear. (In fact, it acts like a gear with only one tooth!)

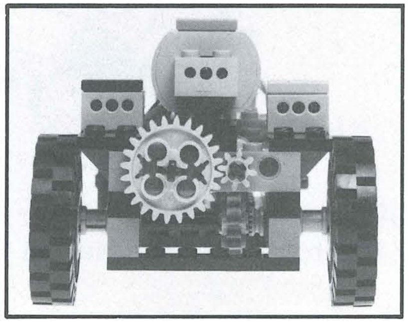


Look at the gear train on your car. Note that every combination of "meshed" gears goes from small to big. Thus, the gear train makes the car much slower and stronger than the motor itself.

Challenge

You can change the gears on the LEGO® car you just built. Here are two possibilities:





For each combination, test how far the car goes in five seconds (use `onfor 50`).

Which one goes the furthest? Why?

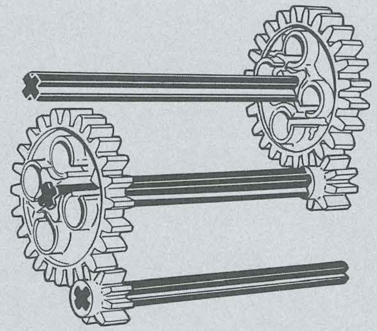
Which one would be best in a tug-of-war? Why?

Challenge

The 8-to-24 combination (the car students originally built from Activity Card 3) should be slower than either the 16-to-16 or the 24-to-8 combinations shown in the photographs. The 8-to-24 combination sacrifices speed for strength. The 24-to-8 combination (the photograph on the right) makes the car go the fastest, so it should go the furthest. The 16-to-16 combination should fall somewhere in between the other two.

The 8-to-24 combination would be the best in a tug of war. It is the slowest, but it has the most strength. This car is also best for climbing hills.

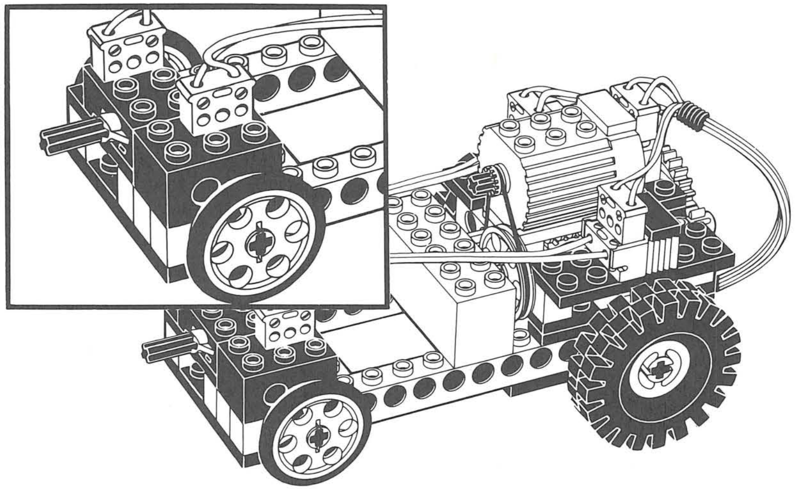
Building gear trains. Students can build gear trains representing different ratios. For example, students can make a 9:1 gear ratio using two 8-tooth gears and two 24-tooth gears.



Counting machines. Students can use gears to make "counting machines." Using 8-tooth gears and 16-tooth gears, you can build a machine that counts in base two. (That is, when you turn one knob four times, the second knob turns twice, and the third knob turns once). Using 8-tooth gears and 24-tooth gears, you can build a machine that counts in base three. Building a machine that counts in base ten is a little more complex, but it can be done.

Sensors in "real" cars. Modern cars use lots of sensors under the hood. Many cars have sensors that measure the temperature of the engine, and other sensors to monitor the mixture of air and fuel in the carburetor. What other types of sensors can you imagine? One idea: a sensor that reports the distance between your car and the car in front of you. If this distance becomes too small, the "car computer" could automatically apply the brakes.

Sensory Car



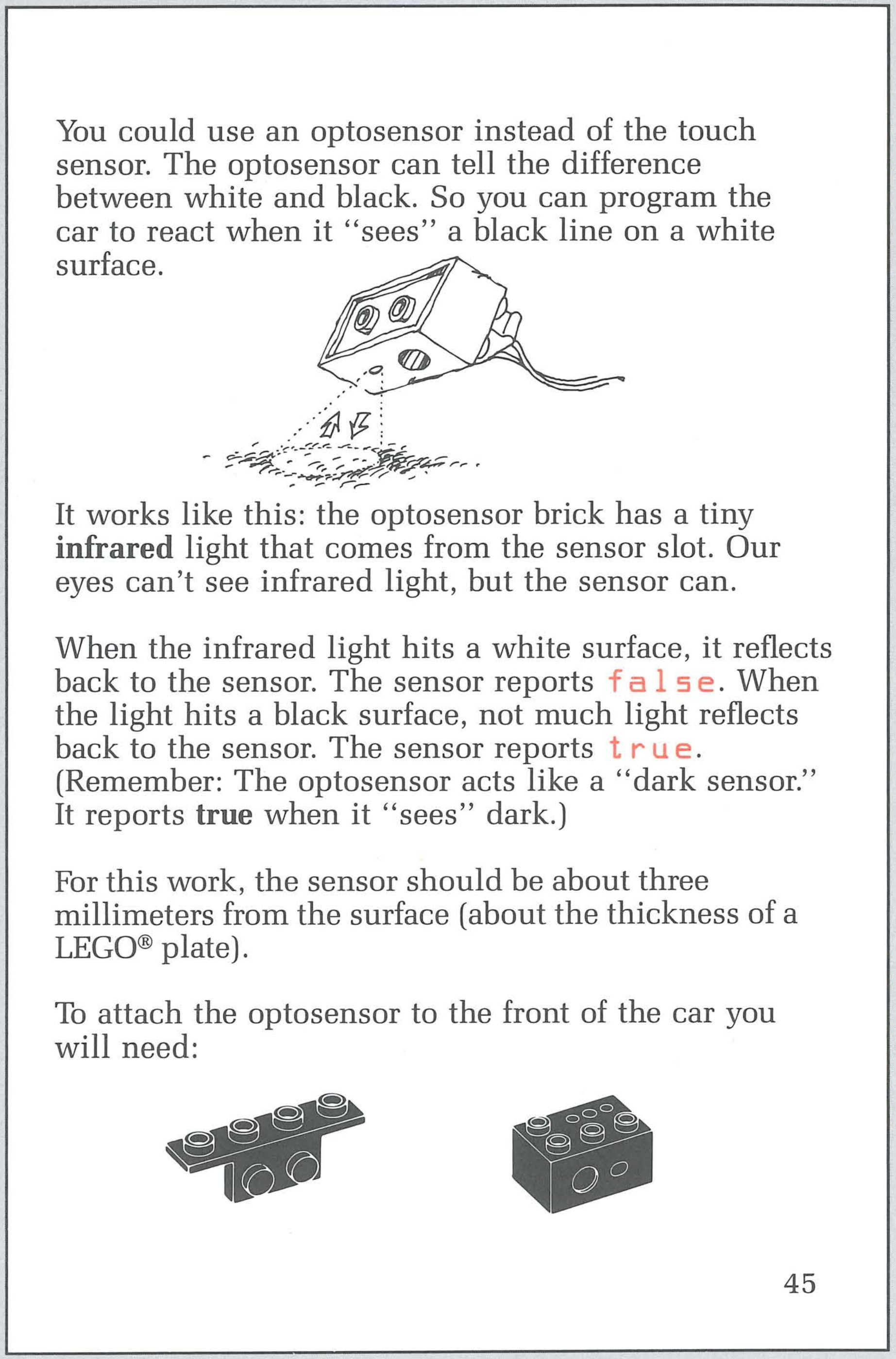
If you add sensors to your car, the car can receive (and react to) information from the world around it.

To start, attach a touch sensor to the front of the car, as shown above. You can write a procedure that makes the car stop when it bumps into something:

```
to collide
talkto "A
on
listento 6
waituntil [sensor?]
off
end
```

Make sure that the motor and sensor are plugged into the correct ports. Then try `collide`.

44



You could use an optosensor instead of the touch sensor. The optosensor can tell the difference between white and black. So you can program the car to react when it "sees" a black line on a white surface.

It works like this: the optosensor brick has a tiny **infrared** light that comes from the sensor slot. Our eyes can't see infrared light, but the sensor can.

You might show where infrared light falls in the spectrum (off the end of the rainbow, past the red).

When the infrared light hits a white surface, it reflects back to the sensor. The sensor reports `false`. When the light hits a black surface, not much light reflects back to the sensor. The sensor reports `true`. (Remember: The optosensor acts like a "dark sensor." It reports **true** when it "sees" dark.)

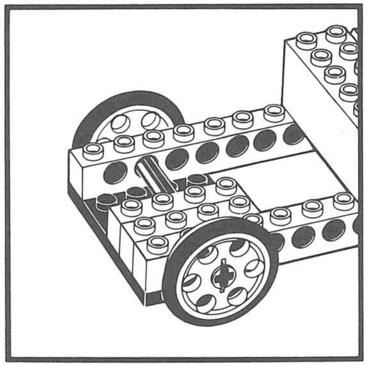
For this work, the sensor should be about three millimeters from the surface (about the thickness of a LEGO® plate).

The optosensor is fairly sensitive to distance. If it is much further than 3 millimeters from the surface (or much closer), it will not work reliably.

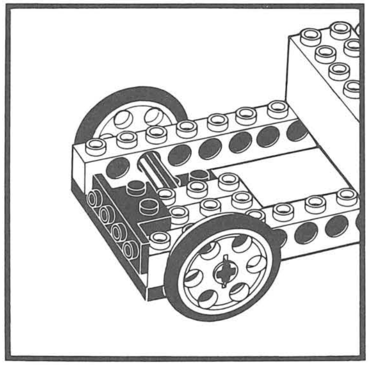
To attach the optosensor to the front of the car you will need:

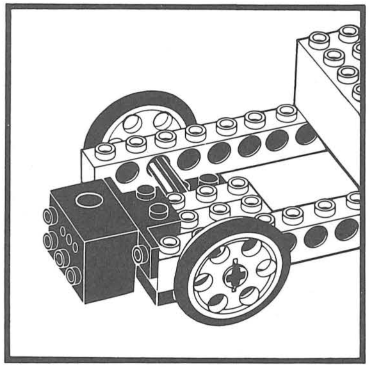
45

Take the optosensor and touch sensor off the front of the car. Also remove the 2 × 4 plate that the touch sensor is attached to. Your car will look like this:



Then build:

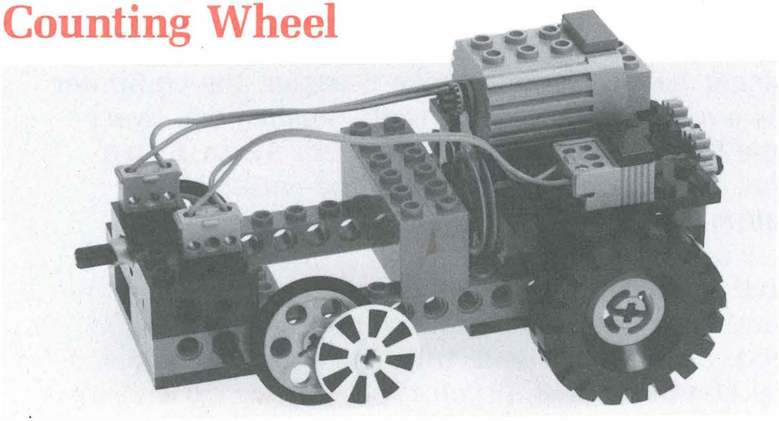




Place a sheet of white paper on the floor. Place a sheet of black paper next to the white sheet, overlapping the edges. Put your car onto the white sheet. Plug the optosensor into port 6. Run the `collide` procedure.

Black lines from magic markers are not as reliable as black construction paper. Black tape will also work. Dull black surfaces work better than shiny ones, since shiny surfaces might reflect too much light.

Counting Wheel



By using an optosensor and a LEGO® counting wheel, you can keep track of exactly how far your car travels.

The counting wheel has black and white slices. On one side, there are four slices of each color. On the other side, there are eight.

When the sensor is next to a black "slice," it reports `true`. When it is next to a white "slice," it reports `false`. Turn `on` the car motor, (type `talkto "A on`), then run the `check` procedure:

```
to check
listento 6
show sensor?
check
end
```

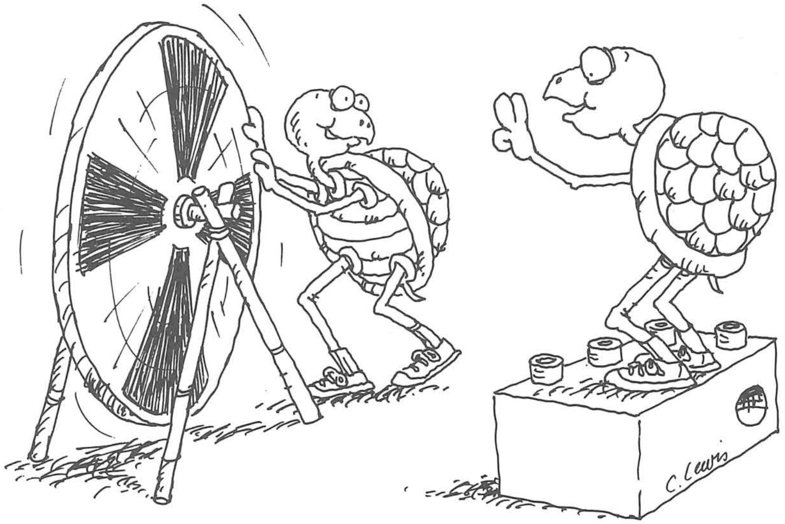
Make sure the sensor is plugged into port 6!
Remember: To stop a recursive procedure, press -S Ctrl-S.

The sensor can count only 16 black/white transitions per second. So the counting wheel is unreliable if it is spinning faster than 2 revolutions per second (or 120 rpm) when the side with 8 black slices is facing the sensor. If the wheel is spinning too quickly, change the gearing of your model.

Actually, it is counting white-to-black transitions. But the number of white-to-black transitions is the same number of white slices (or the number of black slices). A common mistake is to think that the counter will count the **total** number of slices.

You can use this information to keep track of how far the counting wheel has turned — and how far the car has moved! To make it easier, the computer has a special **counter** for each sensor port. Every time the sensor changes from `false` to `true` (that is, from white to black), the counter **automatically** increases by one!

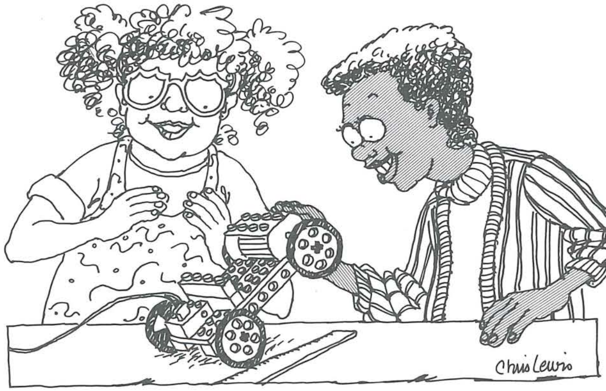
Give it a try. First, type `resetc` in the Command Center to reset the counter to zero. Next, using your hand, turn the counting wheel one complete turn. If you type `show counter` the computer will print the number of white "slices" on the wheel.



Measuring Distance

You can use your car to measure distance. The further the car goes, the higher the counter counts!

First, you must "calibrate" the counter so that you can convert the value of the counter into more familiar units. Type `resetc` to reset the counter to zero. Then, lift up the back of the car and roll the car for exactly one meter (use a ruler). Type `show counter` to find out how high the counter has counted.



Let's assume that the counter has counted to 60. (You will probably get a different number, but the idea will be the same.)

Now you are ready to use the car as a measuring device. Let's say you want to measure the length of a table top. Reset the counter and roll the car from one end of the table to the other. Then check the counter (using `show counter`).

49

Tools to measure distance. People use all sorts of tools to measure distances. There are some obvious examples from everyday experiences: rulers to measure distances in centimeters or inches, car odometers to measure distances in kilometers or miles. But what about very small distances? How might you measure the width of a human hair? And what about large distances? How might you measure the distance to the moon?

Calibration. The process of calibration is important for many types of measurement. For example, when you want to use a scale to weigh something, you want to make sure that the scale reads zero when there is nothing on it. Most scales have an adjustable knob for this purpose.

Another approach to calibration. Here's a very different approach to calibrating a LEGO® car. Assume that the side of the counting wheel with eight slices (four black, four white) is facing the optosensor. So the optosensor will count to four every time the wheel makes one complete revolution.

But every time the wheel makes one complete revolution, the car moves forward a distance equal to the circumference of the wheel. You can calculate the circumference by measuring the diameter of the wheel and multiplying by pi (circumference = pi * diameter = 2 * pi * radius). So the optosensor counts to four while the car moves forward by a distance of pi times the diameter.

If the diameter of the wheel is 2.3 cm., then you can write:

```
to showcm2
listento 6
show 2.3 * 3.14 * counter / 4    ◀pi is about 3.14
showcm2
end
```

Before running this procedure, you should `resetc` the counter and start the motor of the car. How do the readings from `showcm2` compare with the readings from `showcm` on page 50 of *Getting Started*?

Metric measurement. Page 149 of the *Teacher's Guide* provides a copy of three different measuring devices: one in centimeters, one in inches, and one in "LEGO units" (measuring the distance by the number of "studs" on a LEGO brick). The student books use metric measurements in most cases.

If the counter reached 50 in the calibration step, divide by 50 instead of 60.

Challenges

In all of the following procedures, you should `resetc` the counter before beginning the procedure. Also, make sure the computer is "listening to" the correct port.

1. You need a recursive procedure:

```
to showcount
listento 6
show counter
showcount
end
```

2. Assuming a calibration factor of 60:

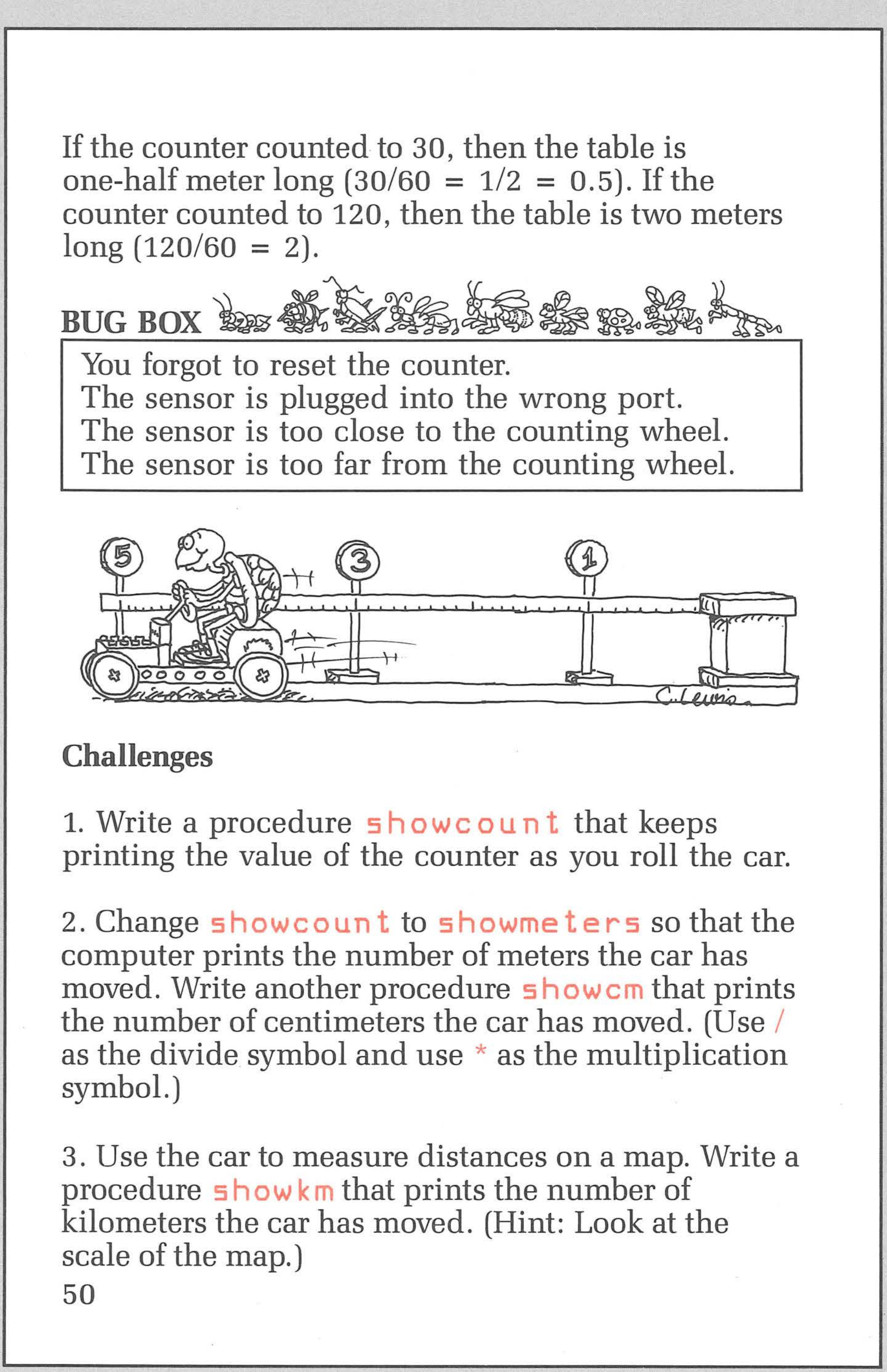
```
to showmeters
listento 6
show counter / 60 ◀You will probably use a different number
showmeters
end

to showcm
listento 6
show (counter / 60) * 100
showcm
end
```

Make sure to leave spaces before and after the `/` and `*` symbols.

3. Assume that the scale of the map is 5 kilometers for every centimeter:

```
to showkm
listento 6
show (counter / 60) * 100 * 5
showkm
end
```

Note that the map measurer car does not have to travel in a straight line. For this reason a non-motorized car would be more useful here. You can measure the length of a coastline, for instance, by rolling your car along the map.



If the counter counted to 30, then the table is one-half meter long (30/60 = 1/2 = 0.5). If the counter counted to 120, then the table is two meters long (120/60 = 2).

BUG BOX

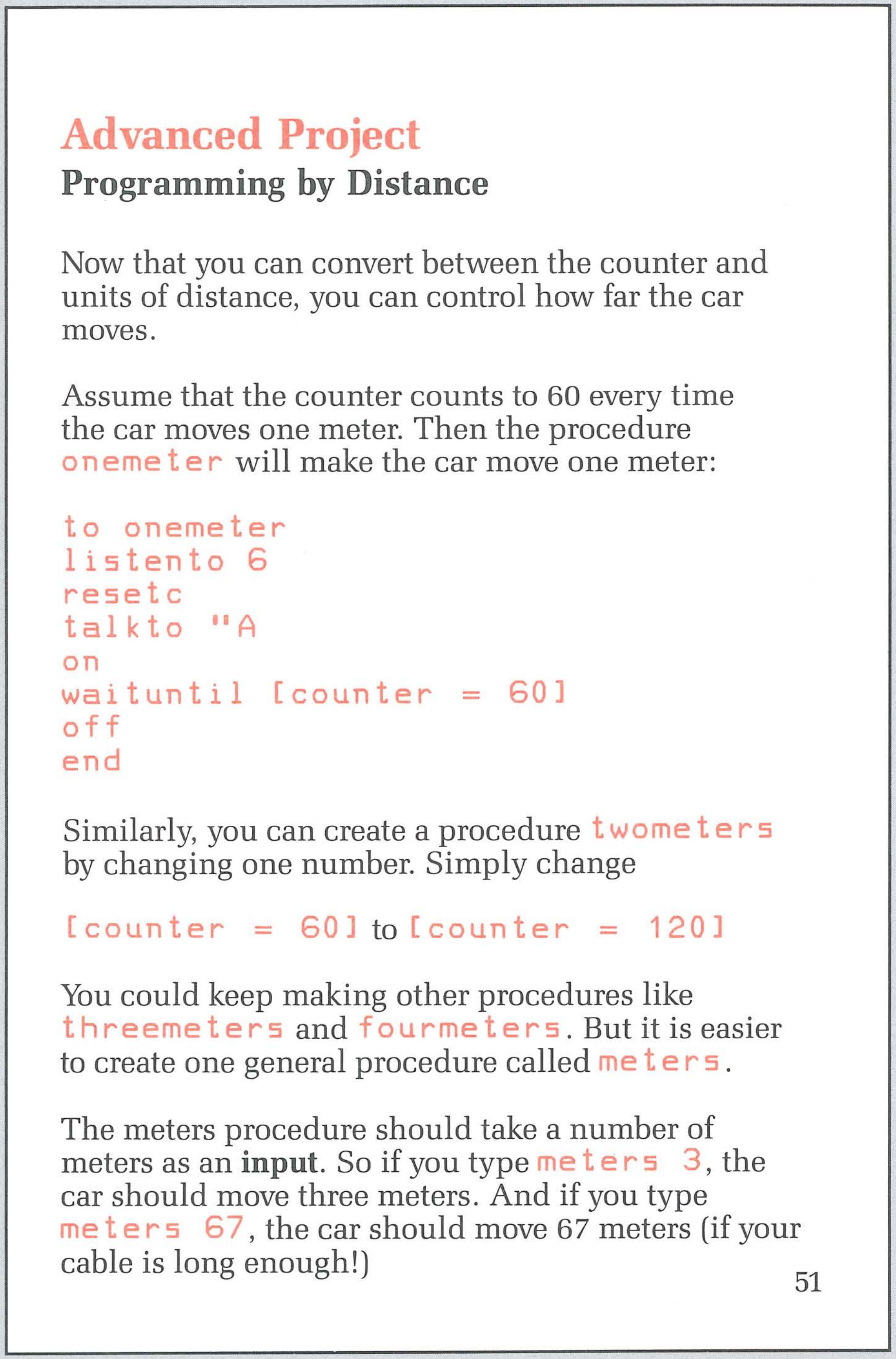
You forgot to reset the counter.
The sensor is plugged into the wrong port.
The sensor is too close to the counting wheel.
The sensor is too far from the counting wheel.

Challenges

1. Write a procedure `showcount` that keeps printing the value of the counter as you roll the car.

2. Change `showcount` to `showmeters` so that the computer prints the number of meters the car has moved. Write another procedure `showcm` that prints the number of centimeters the car has moved. (Use `/` as the divide symbol and use `*` as the multiplication symbol.)

3. Use the car to measure distances on a map. Write a procedure `showkm` that prints the number of kilometers the car has moved. (Hint: Look at the scale of the map.)



Advanced Project

Programming by Distance

Now that you can convert between the counter and units of distance, you can control how far the car moves.

Assume that the counter counts to 60 every time the car moves one meter. Then the procedure `onemeter` will make the car move one meter:

```
to onemeter
listento 6
resetc
talkto "A
on
waituntil [counter = 60]
off
end
```

Similarly, you can create a procedure `twometers` by changing one number. Simply change

`[counter = 60]` to `[counter = 120]`

You could keep making other procedures like `threemeters` and `fourmeters`. But it is easier to create one general procedure called `meters`.

The meters procedure should take a number of meters as an **input**. So if you type `meters 3`, the car should move three meters. And if you type `meters 67`, the car should move 67 meters (if your cable is long enough!)

Distance = Rate * Time. There are many activities involving the relationship between time, rate, and distance. For example, you could change the gearing on a car, then calculate the new speed. You could also use a LEGO® car to "act out" word problems involving time, rate, and distance.

Common Bugs. Whenever you use an expression like `counter = ??`, you must make sure that `??` is an integer (a whole number). For example, the expression `counter = 19.5` is **never** true: `counter` might equal 18 or 19 or 20, but it will never equal 19.5. You can, however, use an expression like `waituntil [counter > 19.5]`. The computer will wait until `counter` is **greater than** 19.5. (That is, it will wait until `counter` reaches 20.)

Also, make sure to put spaces before and after the equal sign. In some versions of Logo, the program will still work without the spaces, but it is good style anyway to surround all arithmetic operations (like `+`, `*`, and `=`) with spaces.

Challenge

Approach #1: Use the fact that `speed = distance / time`. Add two new lines to the `meters` procedure:

```
to meters :distance
listento 6
resetc
resett                                  ◀Reset the timer to
                                         zero
talkto "A
on
waituntil [counter = 60 * :distance]
show :distance / timer                  ◀Calculate the
                                         speed
off
end
```

The procedure simply divides the distance (in meters) by the time (in seconds). The procedure should print the same result no matter what input you use. (If you use a larger input, both the distance and the time will be larger, but the speed should be the same.)

Approach #2:

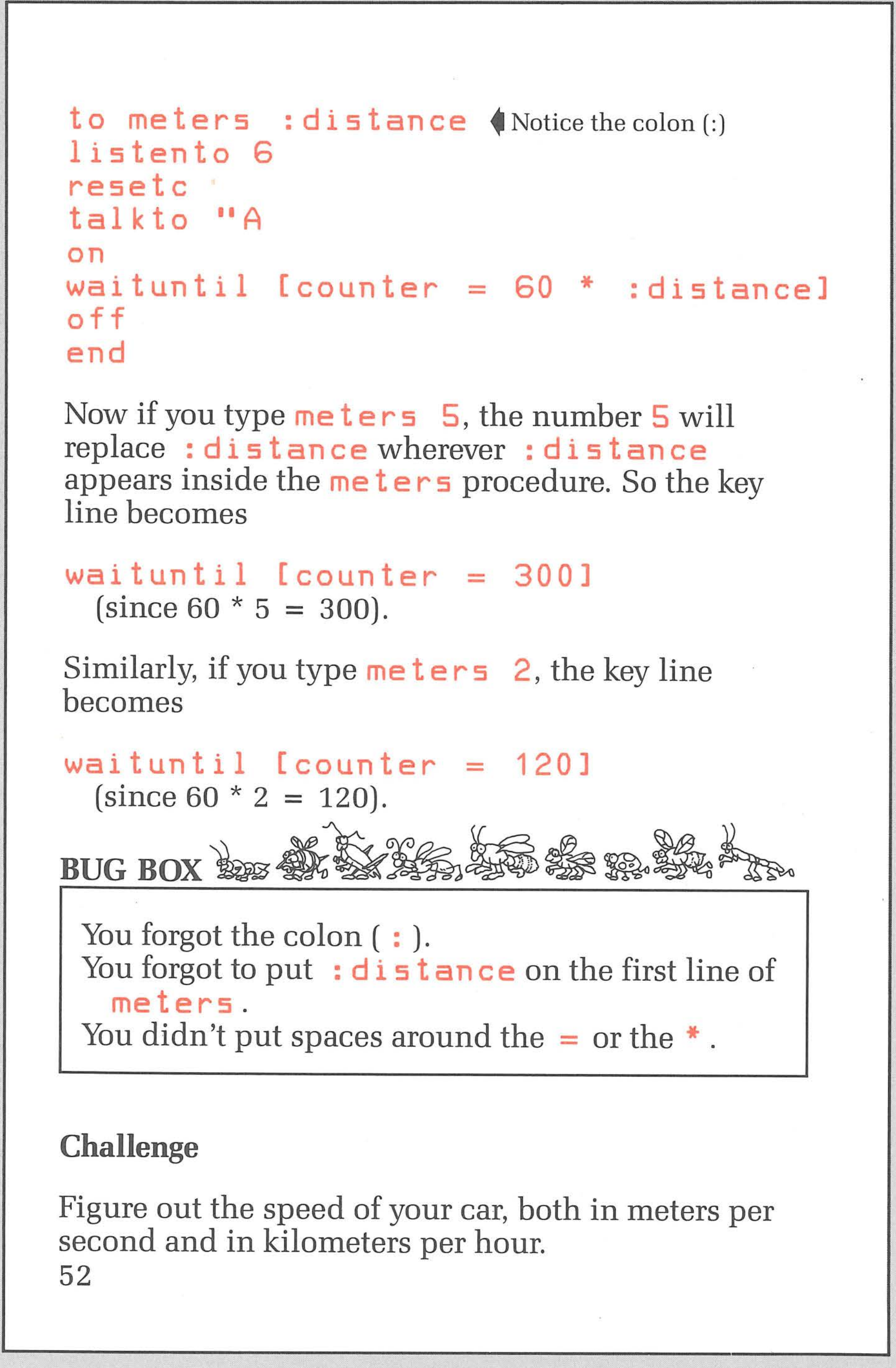
```
to speed :time
listento 6
resetc
talkto "A
onfor :time
show (counter / 60) / (:time / 10)
end
```

This procedure calculates speed by dividing distance in meters (`counter / 60`) by the time in seconds (`:time / 10`). Again, the result should not depend on the input.

To convert meters per second to kilometers per hour, you should multiply by 3600 and divide by 1000 as shown below:

$$\frac{\text{meters}}{\text{second}} \times \frac{\text{3600 seconds}}{\text{1 hour}} \times \frac{\text{1 km}}{\text{1000 meters}} = \frac{\text{km}}{\text{hour}}$$

You can use the computer to do the calculation!



```
to meters :distance     ◀Notice the colon (:)
listento 6
resetc
talkto "A
on
waituntil [counter = 60 * :distance]
off
end
```

Now if you type `meters 5`, the number `5` will replace `:distance` wherever `:distance` appears inside the `meters` procedure. So the key line becomes

```
waituntil [counter = 300]
```

(since 60 * 5 = 300).

Similarly, if you type `meters 2`, the key line becomes

```
waituntil [counter = 120]
```

(since 60 * 2 = 120).

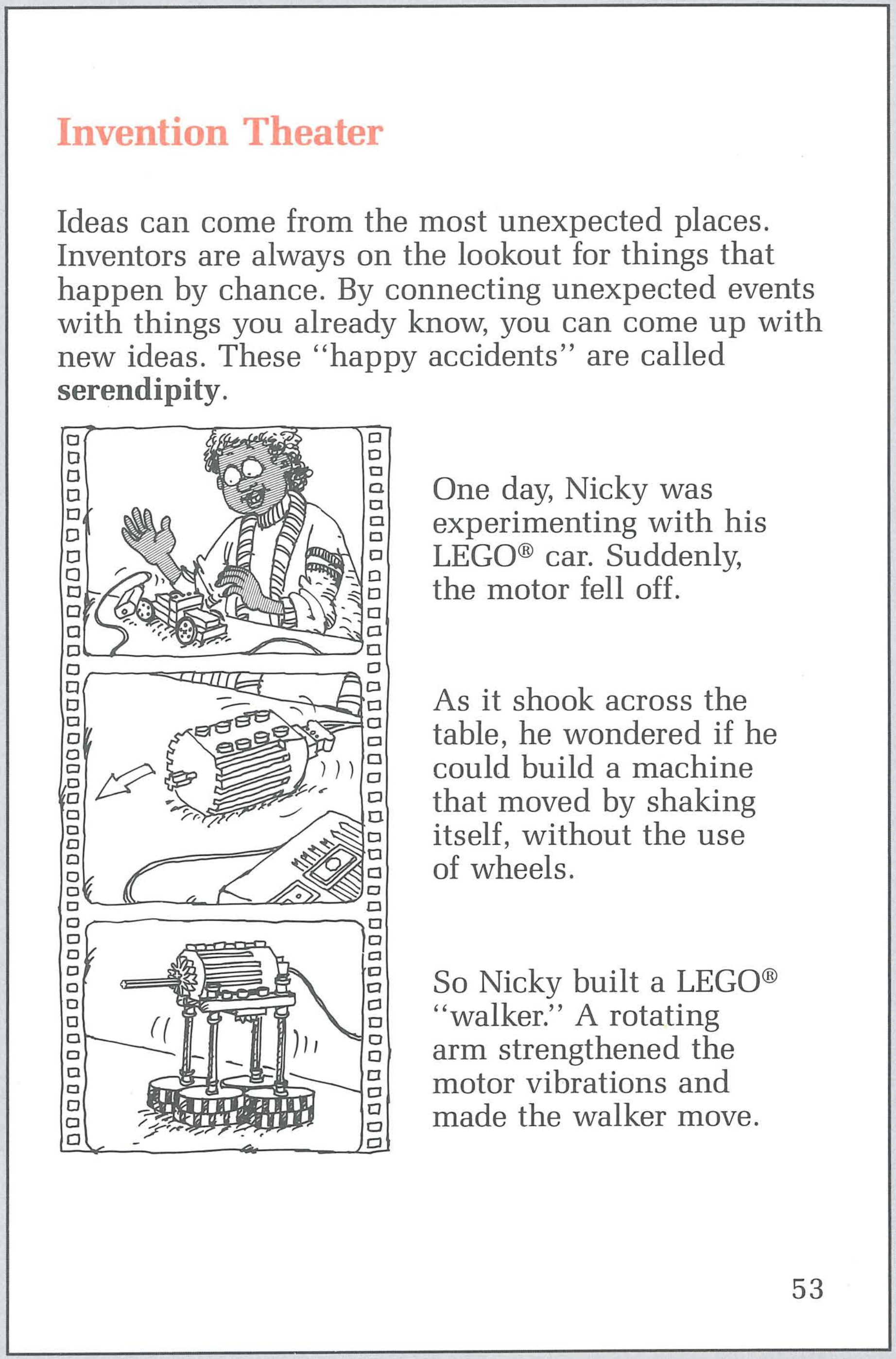
BUG BOX

You forgot the colon (`:`).
You forgot to put `:distance` on the first line of `meters`.
You didn't put spaces around the `=` or the `*`.

Challenge

Figure out the speed of your car, both in meters per second and in kilometers per hour.

52



Invention Theater

Ideas can come from the most unexpected places. Inventors are always on the lookout for things that happen by chance. By connecting unexpected events with things you already know, you can come up with new ideas. These "happy accidents" are called **serendipity**.

One day, Nicky was experimenting with his LEGO® car. Suddenly, the motor fell off.

As it shook across the table, he wondered if he could build a machine that moved by shaking itself, without the use of wheels.

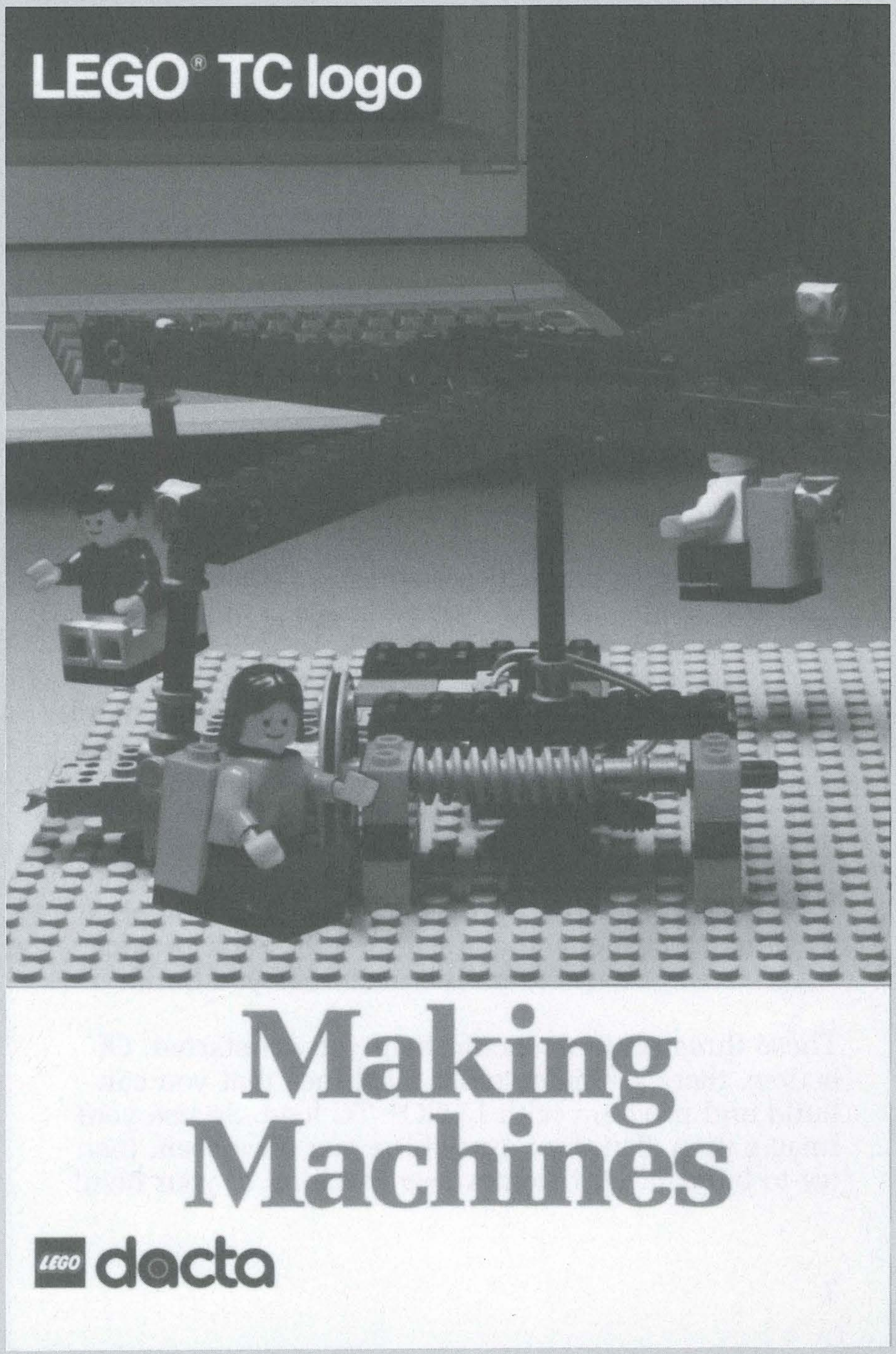
So Nicky built a LEGO® "walker." A rotating arm strengthened the motor vibrations and made the walker move.

It is a good idea to give students plenty of time to work on their own inventions.

You can be an inventor too, just like Nicky.
If you think your invention would make a good story for Invention Theater, write it up and send it to the appropriate address below:

Invention Theater
LEGO Dacta Educational Products
555 Taylor Rd.
Enfield, CT 06082
U.S.A.

Invention Theatre
LEGO Dacta Education Division
P.O. Box 639
Lane Cove
N.S.W. 2066
AUSTRALIA



LEGO® TC logo

Making Machines

LEGO dacta

Chapter 7

Teacher's Notes for *Making Machines*

The *Making Machines* booklet can be used in several different ways:

* **As a source of guided projects.** Three models are discussed: a merry-go-round, a washing machine, and a conveyor belt. The booklet suggests a series of programming projects for each model.

* **As inspiration for "personal" projects.** The booklet suggests three major themes: Machines at Play, Machines at Home, Machines at Work.

Students could design and build their own projects within these theme areas. For example: while one group of students works on the washing machine, others could design and build other machines for an "Automated House." Some possibilities: sewing machines, ovens, toasters.

* **As a focus for social-studies activities.** The models in this booklet touch on many important social-studies themes: the history of invention, the Industrial Revolution, the changing workplace, the impact of computers on society.

Introduction

Machines are everywhere. We use machines at home, at work and at play.

Indeed, our ability to make machines is one thing that makes us different from other animals.

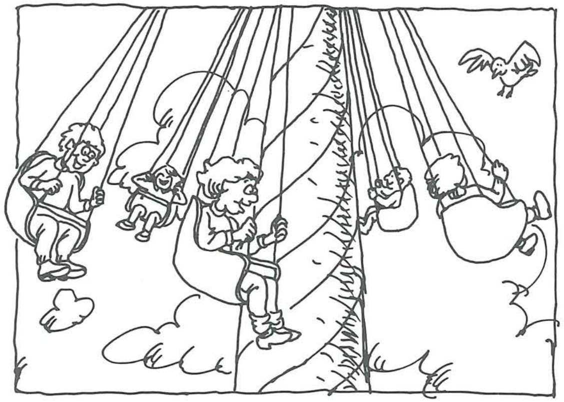
Machines can help us in many ways. Some machines do things that we would find boring or dangerous if we had to do them ourselves.

Other machines **amplify** our abilities. When we use a car, we become faster. When we use a crane, we become stronger.

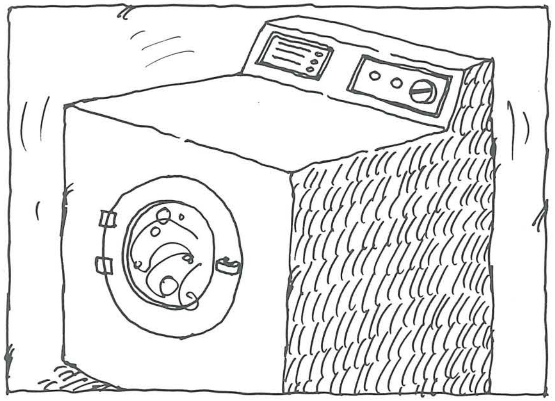
You can use a computer to make machines even more useful. Using a computer, you can **program** a machine to do a complex series of tasks. Then, you can easily change the program to make the machine do something different.

This booklet has three projects: A merry-go-round, a washing machine, and a conveyer belt. Each project suggests ways to program the machines you build.

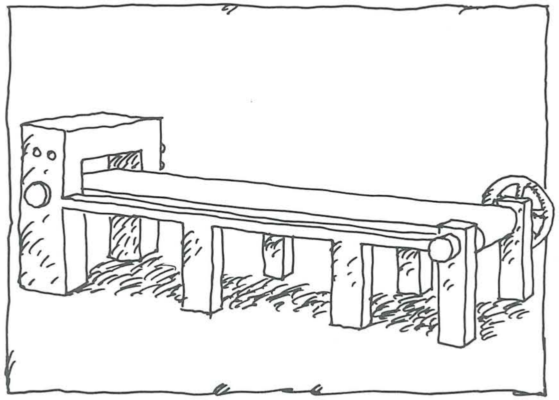
These three projects should help you get started. Of course, there are many other machines that you can build and program with LEGO® TC logo. So use your imagination. Think of a machine you have seen, then try to build it. Or invent a new machine of your own!



Play



Home



Work

Project 1: Merry-Go-Round

Key Ideas

- Themes: Amusement park, machines for entertainment
- Science concepts: Circular motion, centrifugal force
- New programming concept: "Instant" control
- New LEGO® TC logo primitives: `readchar`, `name`

Students could build these other rides instead of (or after) building the merry-go-round. You can discuss the type of motion that characterizes each ride.

Circular motion and centrifugal force. The merry-go-round offers a good example of circular motion. You could discuss other examples of circular motion: a slingshot being twirled; other amusement park rides; the planets revolving around the sun (actually, the planets move in ellipses, not circles); a car turning a corner.

Objects moving in a circle experience centrifugal force. This force seems to "push" objects outward. For example, the swings on the merry-go-round move outward while the ride is spinning around. Children have probably experienced this effect when sitting in a car.

Physicists have a different way of viewing this situation. They say that centrifugal force is only an "apparent" force. When you are in a turning car, your body keeps going forward in a straight line, while the car is turning. The result is that you move toward the outside of the car's circular motion.

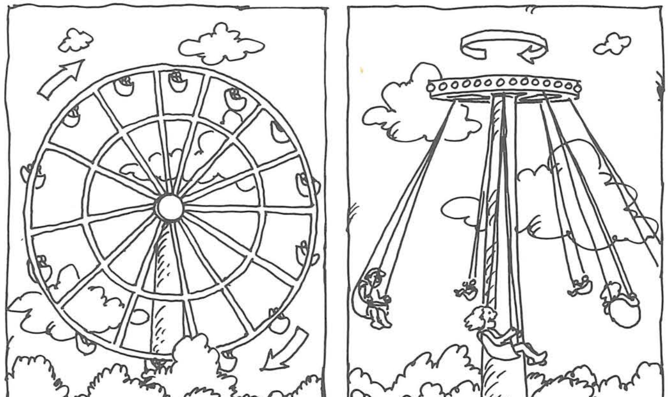
It is useful to discuss **both** ways of viewing the situation. In one view, a force "pushes" you outward. In the other view, you keep going forward and end up on the outside.

Project 1

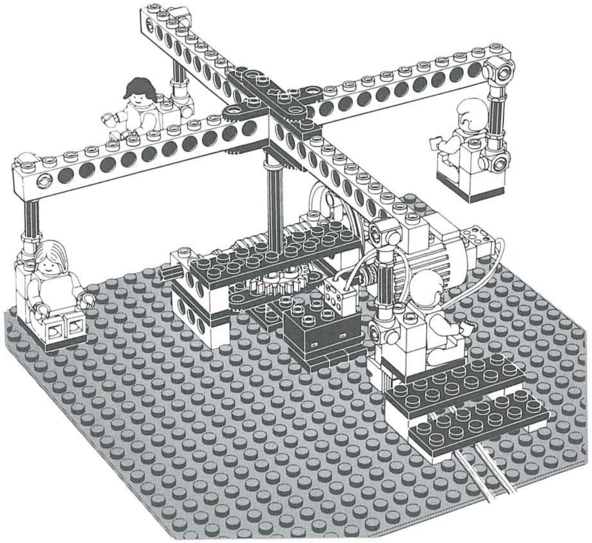
Machines at Play: Merry-Go-Round

Amusement parks are full of machines: Ferris wheels, roller coasters, merry-go-rounds. These machines are fun to ride because of the way they move. Some move you up and down. Others spin you around.

Activity Card 6 shows how to build a LEGO® merry-go-round.



Starting the Ride



Plug the motor into port A, and type `talkto "A`. Now try `onfor 100`. How many times does the merry-go-round go around? How can you make the merry-go-round go around exactly once?

You can change the speed of the merry-go-round using the LEGO® TC logo command `setpower`. Try these commands:

```
on            ◀ Turn on the motor
setpower 3    ◀ Set the motor power to three
setpower 6    ◀ Set the motor power to six
```

To return to "normal" speed, type `setpower 7`.

5

If the merry-go-round revolves five times when you type `onfor 100`, then you should type `onfor 100 / 20` or `onfor 5` to make the merry-go-round go around exactly once.

`setpower` can take any number from 1 to 7.

The greater the power, the greater the speed. That is, the merry-go-round will turn faster after `setpower 6` than after `setpower 3`. But power and speed are not proportional. `setpower 6` will **not** make the merry-go-round turn **twice** as fast as `setpower 3`.

The default for `setpower` is `7`.

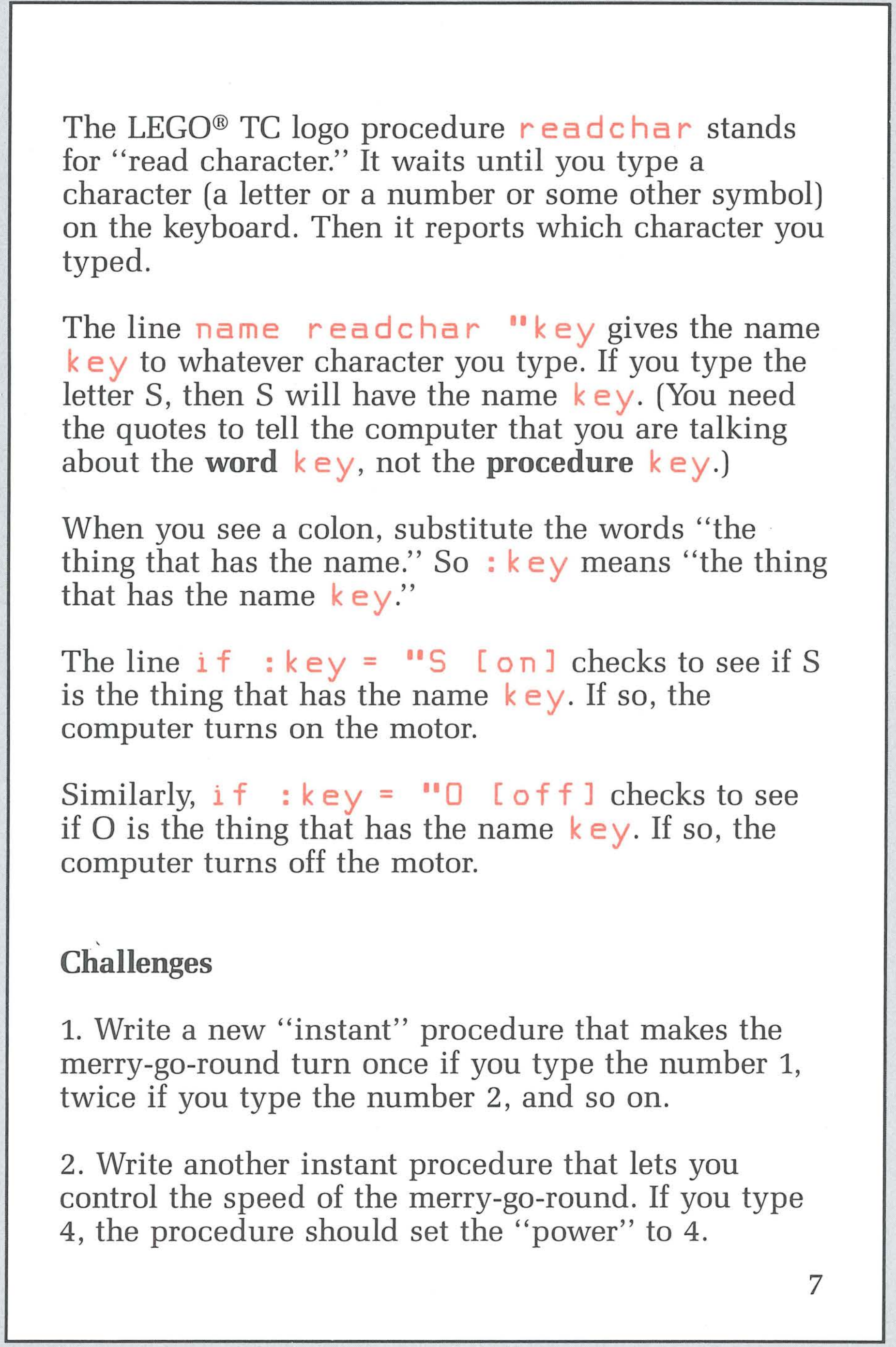
Instant Control

You can write an "instant" procedure to control the merry-go-round. This type of procedure lets you control the machine by pressing a single key on the keyboard.

Let's say that you want the merry-go-round to **S**tart every time you press the S key. And you want the machine to turn **O**ff every time you press the O key, and you want it to **R**everse direction every time you press the R key.

You can write the procedure like this:

```
to instant
talkto "A
name readchar "key    ◀ Don't forget the quotes(")
if :key = "S [on]     ◀ Don't forget the colon (:)
if :key = "O [off]
if :key = "R [rd]
instant
end
```



The LEGO® TC logo procedure `readchar` stands for "read character." It waits until you type a character (a letter or a number or some other symbol) on the keyboard. Then it reports which character you typed.

The line `name readchar "key` gives the name `key` to whatever character you type. If you type the letter S, then S will have the name `key`. (You need the quotes to tell the computer that you are talking about the **word** `key`, not the **procedure** `key`.)

When you see a colon, substitute the words "the thing that has the name." So `:key` means "the thing that has the name `key`."

The line `if :key = "S [on]` checks to see if S is the thing that has the name `key`. If so, the computer turns on the motor.

Similarly, `if :key = "O [off]` checks to see if O is the thing that has the name `key`. If so, the computer turns off the motor.

Challenges

1. Write a new "instant" procedure that makes the merry-go-round turn once if you type the number 1, twice if you type the number 2, and so on.

2. Write another instant procedure that lets you control the speed of the merry-go-round. If you type 4, the procedure should set the "power" to 4.

7

You do **not** even have to hit the Return Enter key.

`readchar` (like `waituntil`) makes the procedure wait until something happens.

It doesn't matter whether you use upper case or lower case when you type S, O, or R.

Remember: To stop a recursive procedure, hold down the or Ctrl key and press the S key (S for stop). If the motor is still on type `alloff` to turn it off.

Challenges

1. Assuming that the merry-go-round revolves once when you type `onfor 20`, you could write the procedure like this:

```
to instant2
talkto "A
name readchar "key
if :key = 1 [onfor 20]
if :key = 2 [onfor 40]
if :key = 3 [onfor 60]
...
...
if :key = 9 [onfor 180]
instant2
end
```

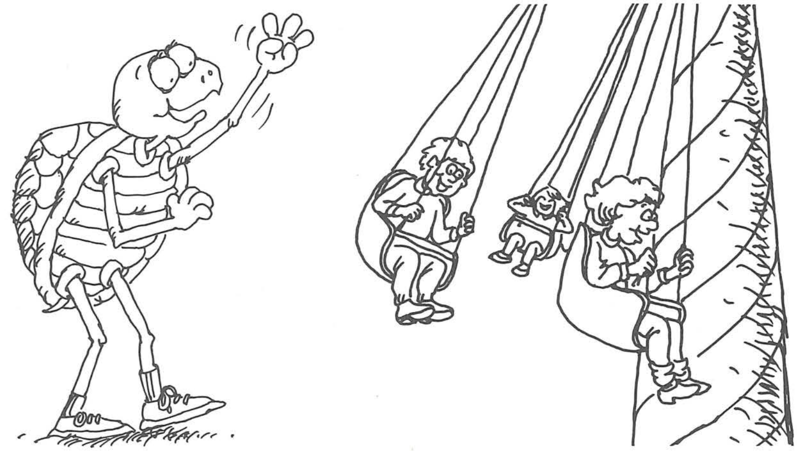
Here is a better approach:

```
to instant2
talkto "A
name readchar "key
onfor 20 * :key
instant2
end
```

If your merry-go-round takes five seconds to make one revolution, then the `onfor` line should be `onfor 50 * :key`.

```
2. to instant3
   talkto "A
   on
   name readchar "key
   setpower :key
   instant3
   end
```

Counting the Turns

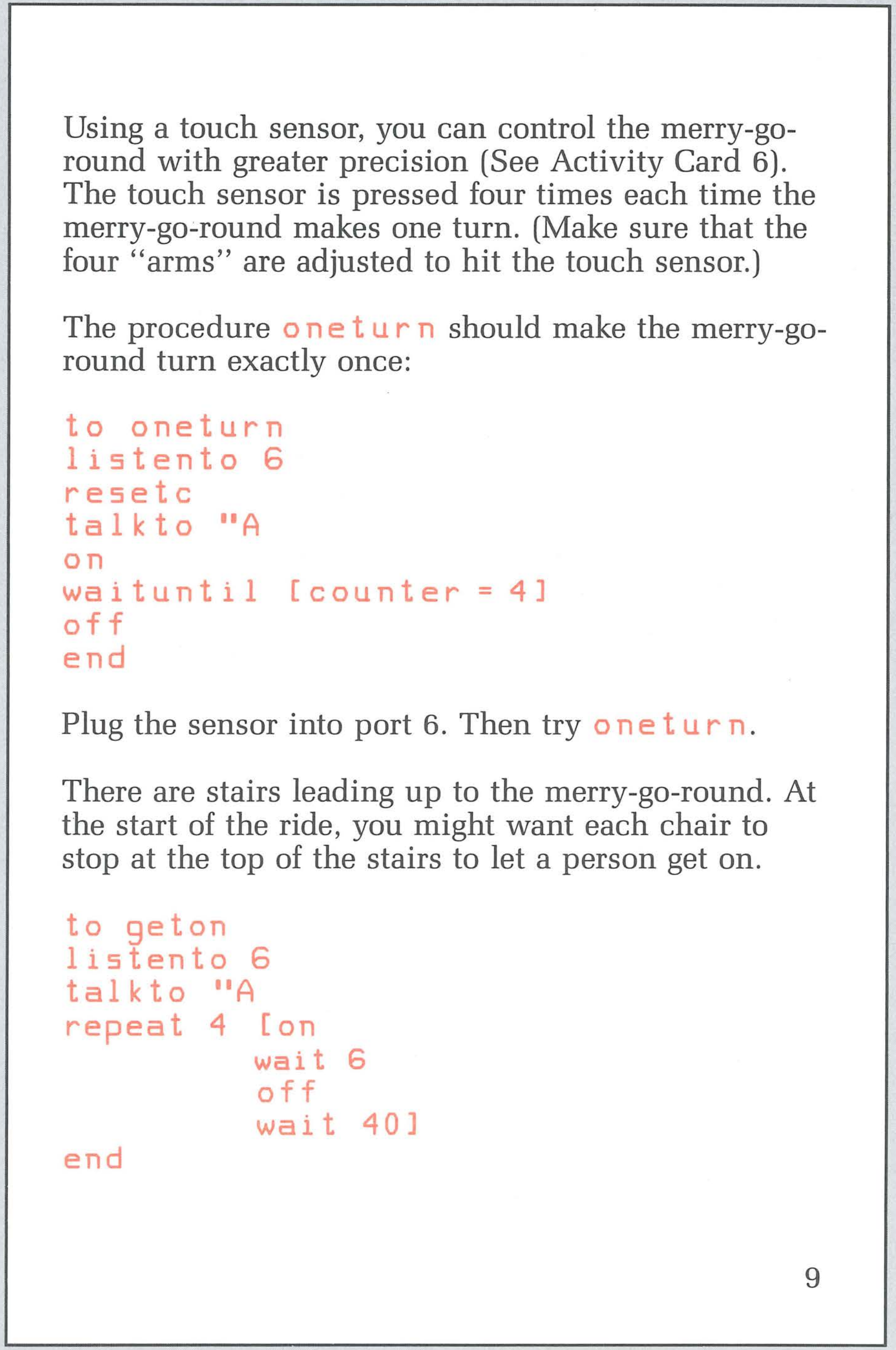


You have already seen how to use `onfor` to control the merry-go-round.

There is a problem with this approach. Let's say that the merry-go-round revolves exactly once when you type `onfor 10`. What will happen the next time you type `onfor 10`? Will the merry-go-round revolve exactly once?

Maybe, but maybe not. Imagine that a LEGO® piece has slipped a bit, increasing the friction in the system. When you type `onfor 10` again, the merry-go-round will not go as far as before.

Another problem: Some motors work ''better'' in one direction than the other. So `onfor 10` in one direction is not the same as `onfor 10` in the other direction.



Using a touch sensor, you can control the merry-go-round with greater precision (See Activity Card 6). The touch sensor is pressed four times each time the merry-go-round makes one turn. (Make sure that the four “arms” are adjusted to hit the touch sensor.)

The procedure `oneturn` should make the merry-go-round turn exactly once:

```
to oneturn
listento 6
resetc
talkto "A
on
waituntil [counter = 4]
off
end
```

Plug the sensor into port 6. Then try `oneturn`.

There are stairs leading up to the merry-go-round. At the start of the ride, you might want each chair to stop at the top of the stairs to let a person get on.

```
to geton
listento 6
talkto "A
repeat 4 [on
         wait 6
         off
         wait 40]
end
```

The `geton` procedure is formatted to make it easy to read. You can format procedures any way you like. Exceptions: The procedure label (starting with `to`) and the `end` statement must be alone on their lines.

Challenges

1. It is best to make use of subprocedures:

```
to ride :revolutions
geton
turns :revolutions
getoff
end

to getoff
geton
end
```

Note that the definition for `ride` uses `turns :revolutions`, since `:revolutions` is the input to the `ride` procedure. It doesn't matter that `:number` was used as the variable name in the definition of `turns`.

The `ride` procedure would work just as well if you used `geton` in place of `getoff` (since they both do the same thing). But it is good practice to give procedures meaningful names.

2. Here is one approach:

```
to turns :number
listento 6
resetc
talkto "A
on
waituntil [counter = 4 * :number]
off
end
```

For a review of writing procedures with inputs, see page 52 of *Getting Started*.

3. If you put the side of the counting wheel with 16 thin "slices" facing the sensor, the counter will count up to eight every time the merry-go-round revolves once. So in `oneturn`, you must change `waituntil [counter = 4]` to `waituntil [counter = 8]`.

BUG BOX

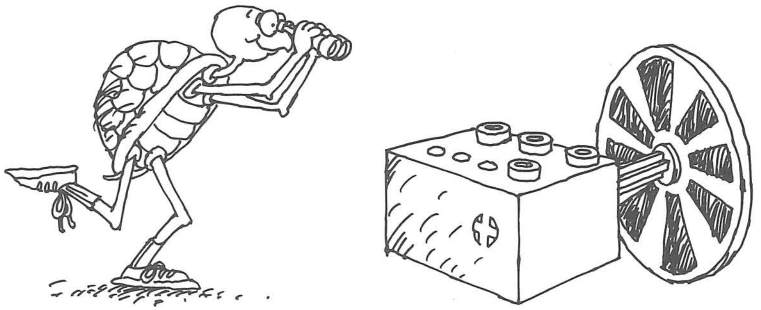
The wrong side of the wheel is facing the sensor.
You left a space in `oneturn` or `geton`.

Challenges

1. Write a procedure `ride` that controls the entire ride. Here's one approach: First, the merry-go-round stops at each chair to let people get on. Then it turns a given number of times. Finally it stops at each chair to let people get off.

2. Write a procedure `turns` that takes a number as an input. If you type `turns 4`, the merry-go-round should go around four times. (Hint: See page 52 of *Getting Started*.)

3. Replace the touch sensor with an optosensor and counting wheel, as shown on the back page of Activity Card 6. Change the `oneturn` procedure so that it works with the optosensor.



10

Merry-Go-Round Music

You can use the LEGO® TC logo primitive `tone` to make the computer play music while the merry-go-round turns.

Here's an example:

```
to merry
talkto "A
on
tone 330   9
tone 311   3
tone 330   3
tone 392   3
tone 523  12
tone 494   3
tone 523   3
tone 587   3
tone 523   3
tone 494   3
tone 523   3
tone 330   3
tone 392   3
tone 494  12
wait 9
merry
end
```

The first input to `tone` sets the **pitch** of the tone. The second input sets the **duration** of the tone.

Try changing the `tone` commands to make a new song for the `merry` procedure.

11

Note that `merry` is recursive, so the music will keep playing over and over. To stop the procedure, press -S Ctrl-S. To stop the merry-go-round, type `off`.

The frequencies of other pitches are listed under the `tone` definition in the *Reference Guide*.

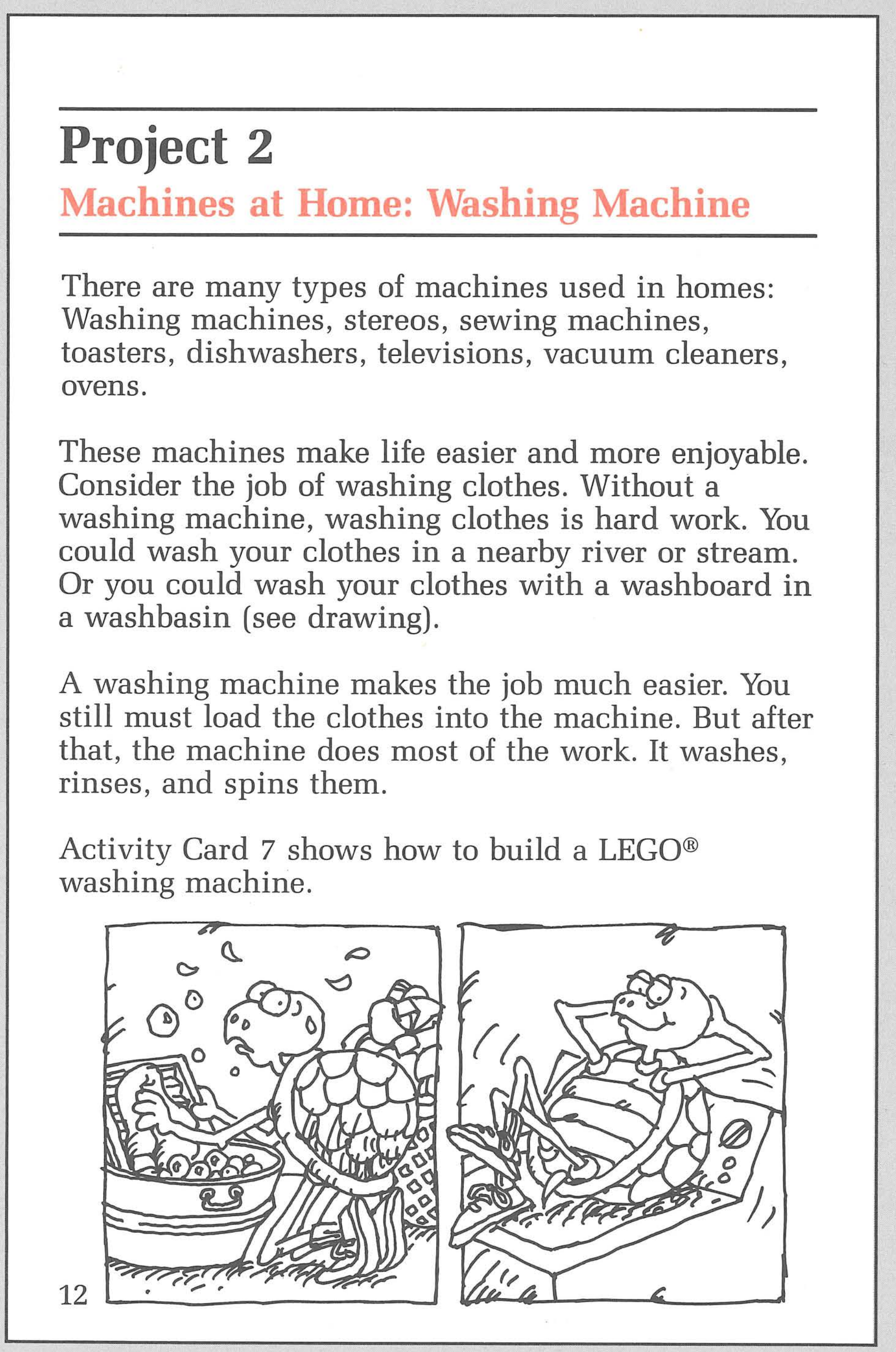
Project 2: Washing Machine

Key Ideas

- Themes: Machines in the Home, the Automated House, House of the Future
- Use of indicator lights
- Importance of safety in machine design
- Advanced use of the LEGO® timer
- New LEGO® TC logo primitive: `or`, `output`

Students could build these other machines after (or instead of) building the washing machine.

History of household chores. You can discuss how household chores have changed through history. Some household chores that were common in the past (like making butter) are no longer common. People now buy many products that they used to make themselves. People still do other chores (like washing clothes), but machines have changed how the chores are done. You might make a list of household chores in the year 1776 (or 1788 or some other year) and another list of household chores today. How are the lists different?



Project 2

Machines at Home: Washing Machine

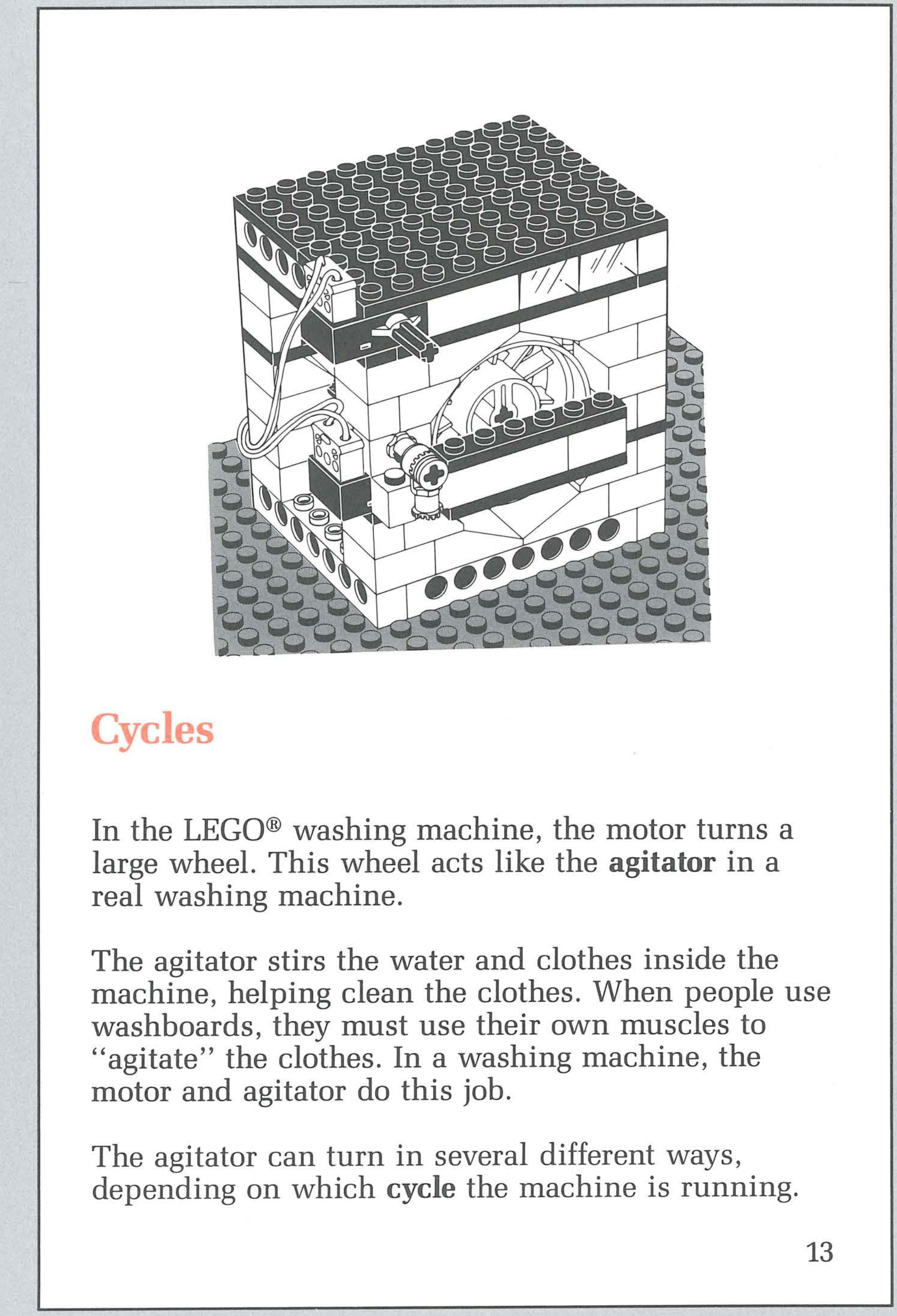
There are many types of machines used in homes: Washing machines, stereos, sewing machines, toasters, dishwashers, televisions, vacuum cleaners, ovens.

These machines make life easier and more enjoyable. Consider the job of washing clothes. Without a washing machine, washing clothes is hard work. You could wash your clothes in a nearby river or stream. Or you could wash your clothes with a washboard in a washbasin (see drawing).

A washing machine makes the job much easier. You still must load the clothes into the machine. But after that, the machine does most of the work. It washes, rinses, and spins them.

Activity Card 7 shows how to build a LEGO® washing machine.

12



Cycles

In the LEGO® washing machine, the motor turns a large wheel. This wheel acts like the **agitator** in a real washing machine.

The agitator stirs the water and clothes inside the machine, helping clean the clothes. When people use washboards, they must use their own muscles to "agitate" the clothes. In a washing machine, the motor and agitator do this job.

The agitator can turn in several different ways, depending on which **cycle** the machine is running.

13

In top-loading washing machines, the agitator is vertical.

Real washing machines have more cycles. You could discuss other types of cycles.

Challenges

1. Here's one approach:

```
to wash
talkto "B
repeat 15 [onfor 2 rd]
end
```

2. To make the agitator spin for ten seconds:

```
to spin
talkto "B
onfor 100
end
```

3. This is simple if you use subprocedures:

```
to laundry
wash
spin
end
```

In the **wash** cycle, the agitator turns back and forth. In the **spin-dry** cycle, the agitator turns in just one direction.

Here's a procedure for the wash cycle:

```
to wash
talkto "B
repeat 15 [onfor 5 rd]
end
```

Make sure the motor is plugged into port B. Then try running `wash`.

BUG BOX

The motor is not plugged in.
You are "talking" to the wrong port.
The stop button is pushed in.

Challenges

1. Change the `wash` procedure so that the agitator turns back and forth more quickly.

2. Write a procedure called `spin` for the spin-dry cycle of the washing machine. The agitator should simply spin in one direction.

3. Write a procedure called `laundry` that runs a wash cycle, then a spin-dry cycle. (Hint: Use subprocedures.)

Indicator Lights

Lights can be very useful for displaying information. On a washing machine, lights can show which cycle is running.

The LEGO® washing machine has two lights. You can change the `wash` procedure so that one of the lights is on during the wash cycle. Plug a light into port 0 and a motor into port B.

```
to wash
talkto 0
on
talkto "B
repeat 15 [onfor 5 rd]
talkto 0
off
end
```

Now the light plugged into port 0 will turn on whenever you run `wash`.

Challenge

Change the `spin` procedure so that the other light is on during the spin-dry cycle. Do you have to change anything in the `laundry` procedure?

The traffic light is another example where lights display information.

You could use `talkto [B 0]` to talk to both the light and the motor. But the light will flash briefly every time the motor changes direction. (The motor turns off briefly when it changes direction.)

Challenge

Assuming the other light is in port 1.

```
to spin
talkto [B 1]
onfor 100
end
```

Note that the procedure talks to the motor and the light together, since they are always on together.

You don't have to change the `laundry` procedure at all. All of the necessary changes are in the subprocedures. This is an example of good program design: changes at one level should not necessitate changes at other levels.

When you name things, it is much easier for other people to understand your programs. Also, you are less likely to get confused yourself. Naming things becomes more important as programs become more complex.

Students often get confused by the quotes and colons involved in naming. See page 12 of *Getting Started* for an explanation of quotes. See page 52 of *Getting Started* for an explanation of colons.

Starter Switch

The washing machine has two touch sensors. The one next to the indicator lights acts as a **starter switch**. The one next to the door acts as a **safety sensor**.

To help remember which sensor is in which port, you can give names to the ports. Plug the starter switch in port 6 and the safety sensor in port 7. Then type in the Command Center:

```
name 6 "starter
name 7 "safety
```

Now `listento :starter` is the same as `listento 6`. Giving names to ports is particularly useful when you use many ports at the same time.

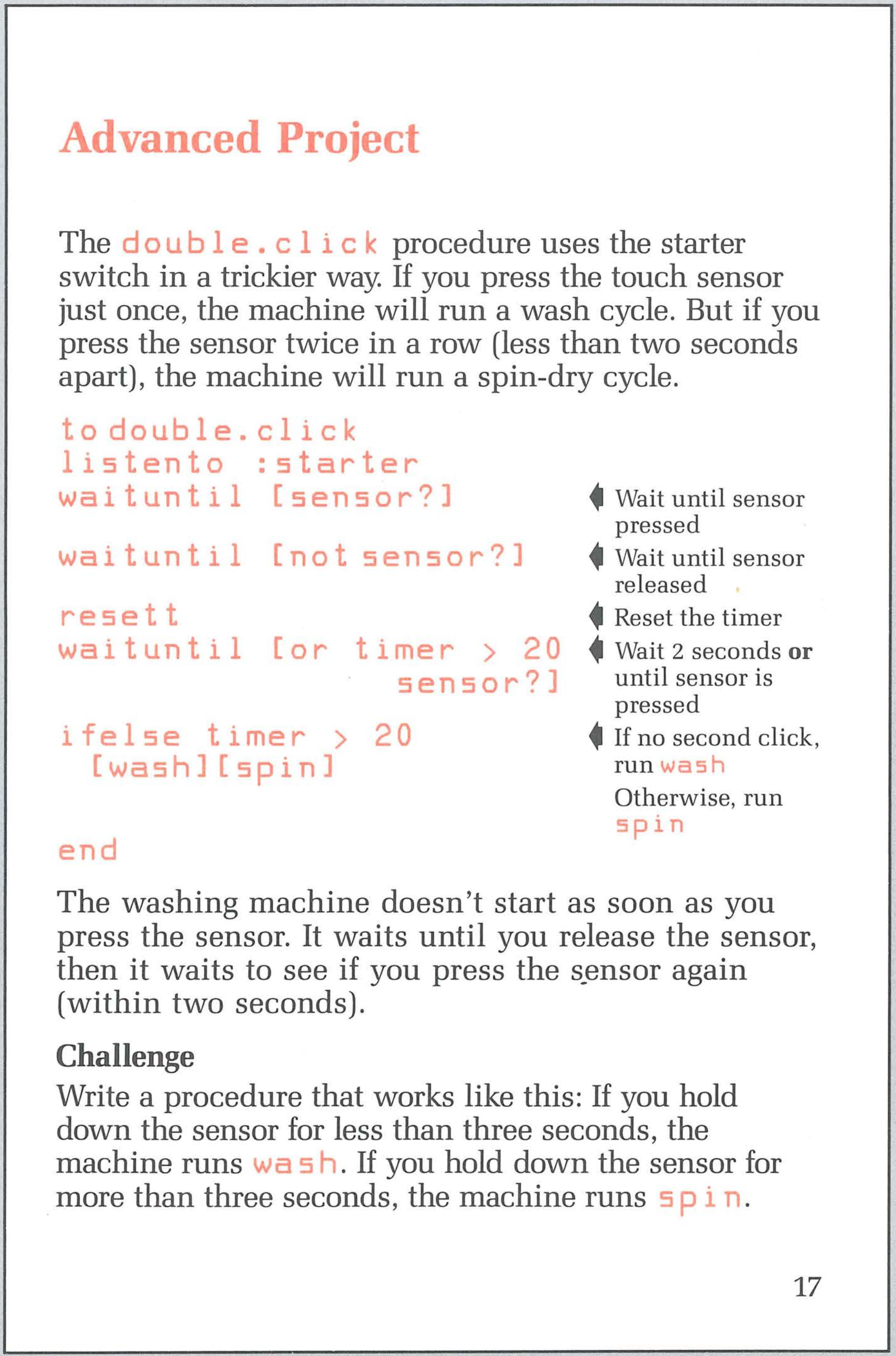
To use the touch sensor as a starter switch, type this procedure:

```
to start
listento :starter
waituntil [sensor?]
laundry
end
```

Run the `start` procedure. The computer waits until you press the touch sensor, then it runs the procedure `laundry`.

BUG BOX

You forgot the quotes(") in the `name` command.
You wrote `listento "starter` instead of `:starter`.



Advanced Project

The `double.click` procedure uses the starter switch in a trickier way. If you press the touch sensor just once, the machine will run a wash cycle. But if you press the sensor twice in a row (less than two seconds apart), the machine will run a spin-dry cycle.

```
to double.click
listento :starter
waituntil [sensor?]              Wait until sensor
                                 pressed
waituntil [not sensor?]          Wait until sensor
                                 released
resett                           Reset the timer
waituntil [or timer > 20         Wait 2 seconds or
              sensor?]           until sensor is
                                 pressed
ifelse timer > 20                If no second click,
  [wash][spin]                   run wash
                                 Otherwise, run
                                 spin
end
```

The washing machine doesn't start as soon as you press the sensor. It waits until you release the sensor, then it waits to see if you press the sensor again (within two seconds).

Challenge

Write a procedure that works like this: If you hold down the sensor for less than three seconds, the machine runs `wash`. If you hold down the sensor for more than three seconds, the machine runs `spin`.

17

If you have used a Macintosh™ computer: the Macintosh™ mouse has this same capability. A "double click" has a different meaning than a "single click."

`or` can be very useful. It reports `true` if **any** of the statements that follow are true. It reports `false` if **all** of the statements are false.

There is also a LEGO® TC logo primitive `and`. It reports `true` if **all** of the statements that follow are true. It reports `false` if **any** of the statements are false.

You can adjust the timing so that the computer waits 4 seconds (use `timer > 40`). This allows "slower" double clicks. Or you could adjust it to half a second (`timer > 5`). This requires a "faster" double click.

Remember: The timer counts in tenths of seconds.

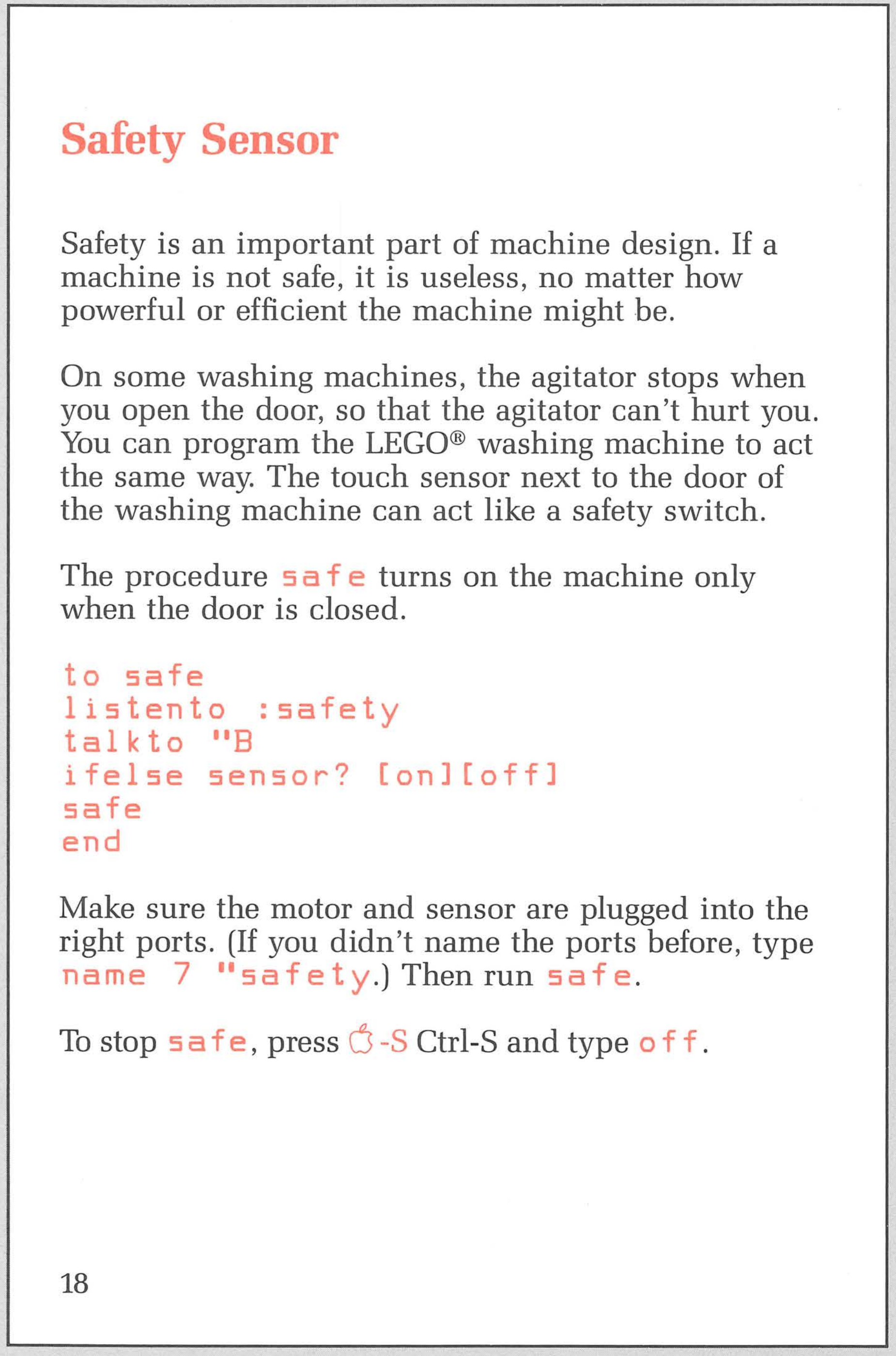
Challenge

Let's call the new procedure `timed.button`. The difference between `double.click` and `timed.button` is somewhat subtle. `timed.button` pays attention to how long you hold the sensor down, while `double.click` pays attention to how long you **don't** hold it down (between clicks).

```
to timed.button
listento :starter
waituntil [sensor?]            Wait for the person to press
resett
waituntil [not sensor?]        Wait for the person to release
ifelse timer > 30 [spin][wash]
end
```

Morse code. The `timed.button` procedure can tell the difference between "long" pushes of the touch sensor and "short" pushes. You could extend this idea to make a Morse-code procedure. The computer should interpret short pushes as "dots" and long pushes as "dashes." For example, if you give the sensor one short push (a dot), the computer should print the letter E. If you give it one short push then one long one (dot-dash), the computer should print the letter A. (Note: This Activity has recently been developed by LEGO Dacta Educational Products. Contact them for more information.)

Product safety. The safety sensor on the washing machine door could lead to a discussion of product safety. You might discuss how the "consumer movement" has pushed for improvements in product safety. You could use the automobile as an example. The government now requires cars to have stronger bumpers, and it requires all cars to have seat belts. Students might try to think of other ways to improve product safety.



Safety Sensor

Safety is an important part of machine design. If a machine is not safe, it is useless, no matter how powerful or efficient the machine might be.

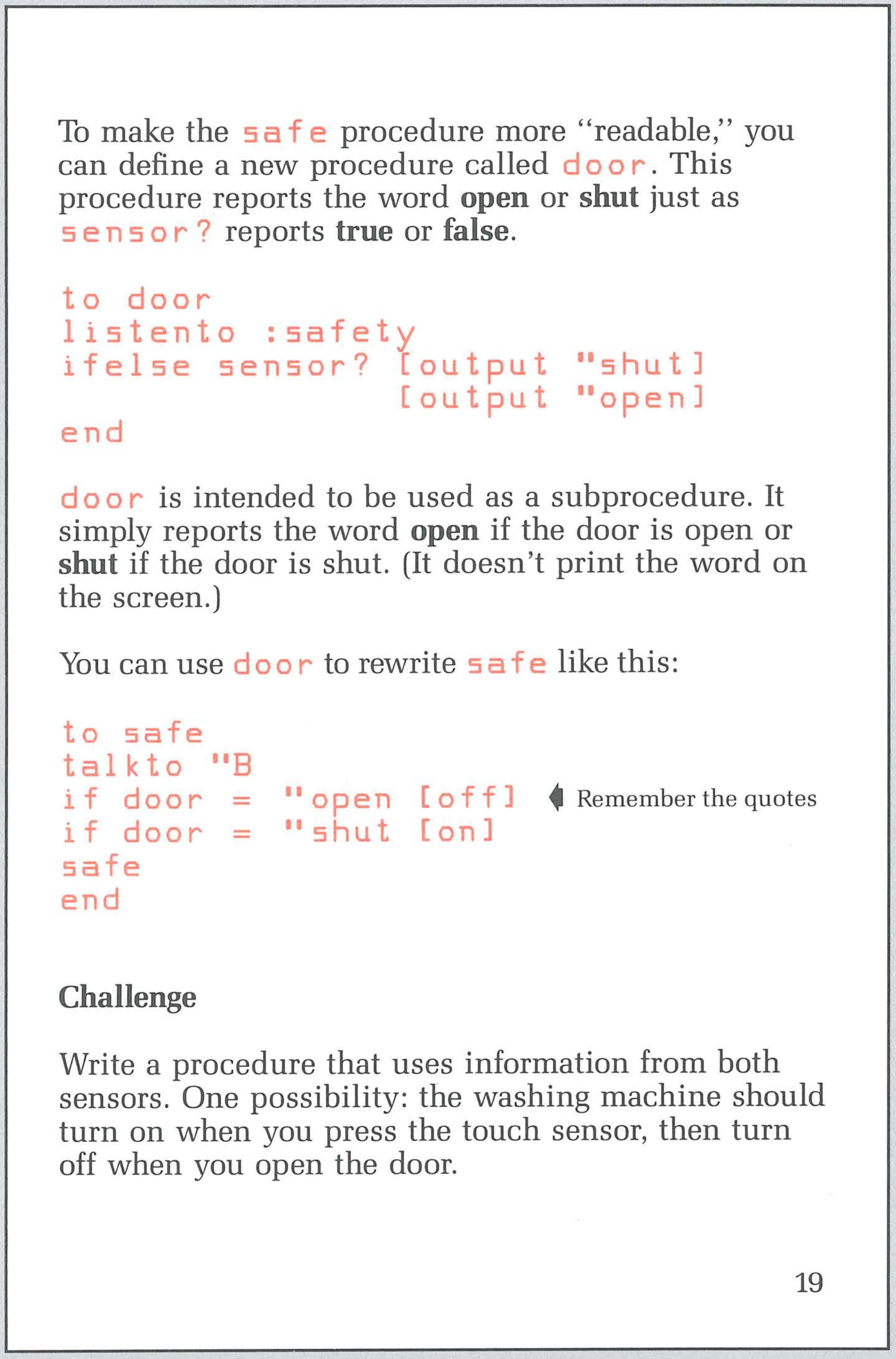
On some washing machines, the agitator stops when you open the door, so that the agitator can't hurt you. You can program the LEGO® washing machine to act the same way. The touch sensor next to the door of the washing machine can act like a safety switch.

The procedure `safe` turns on the machine only when the door is closed.

```
to safe
listento :safety
talkto "B
ifelse sensor? [on][off]
safe
end
```

Make sure the motor and sensor are plugged into the right ports. (If you didn't name the ports before, type `name 7 "safety`.) Then run `safe`.

To stop `safe`, press -S Ctrl-S and type `off`.



To make the `safe` procedure more "readable," you can define a new procedure called `door`. This procedure reports the word **open** or **shut** just as `sensor?` reports **true** or **false**.

```
to door
listento :safety
ifelse sensor? [output "shut]
               [output "open]
end
```

`door` is intended to be used as a subprocedure. It simply reports the word **open** if the door is open or **shut** if the door is shut. (It doesn't print the word on the screen.)

You can use `door` to rewrite `safe` like this:

```
to safe
talkto "B
if door = "open [off]
if door = "shut [on]
safe
end
```

◀ Remember the quotes

Challenge

Write a procedure that uses information from both sensors. One possibility: the washing machine should turn on when you press the touch sensor, then turn off when you open the door.

The LEGO® TC logo primitive `output` tells the computer to **report** whatever follows. So `door` reports the word `open` or the word `shut`, just as `sensor?` reports either `true` or `false`.

`safe` works fine without the use of the `door` procedure. But it is a good idea to make procedures as "readable" and understandable as possible. This is particularly important when you want to share your procedures with other people (perhaps as part of a group project) and when you want to debug or change a procedure at a later time.

Challenge

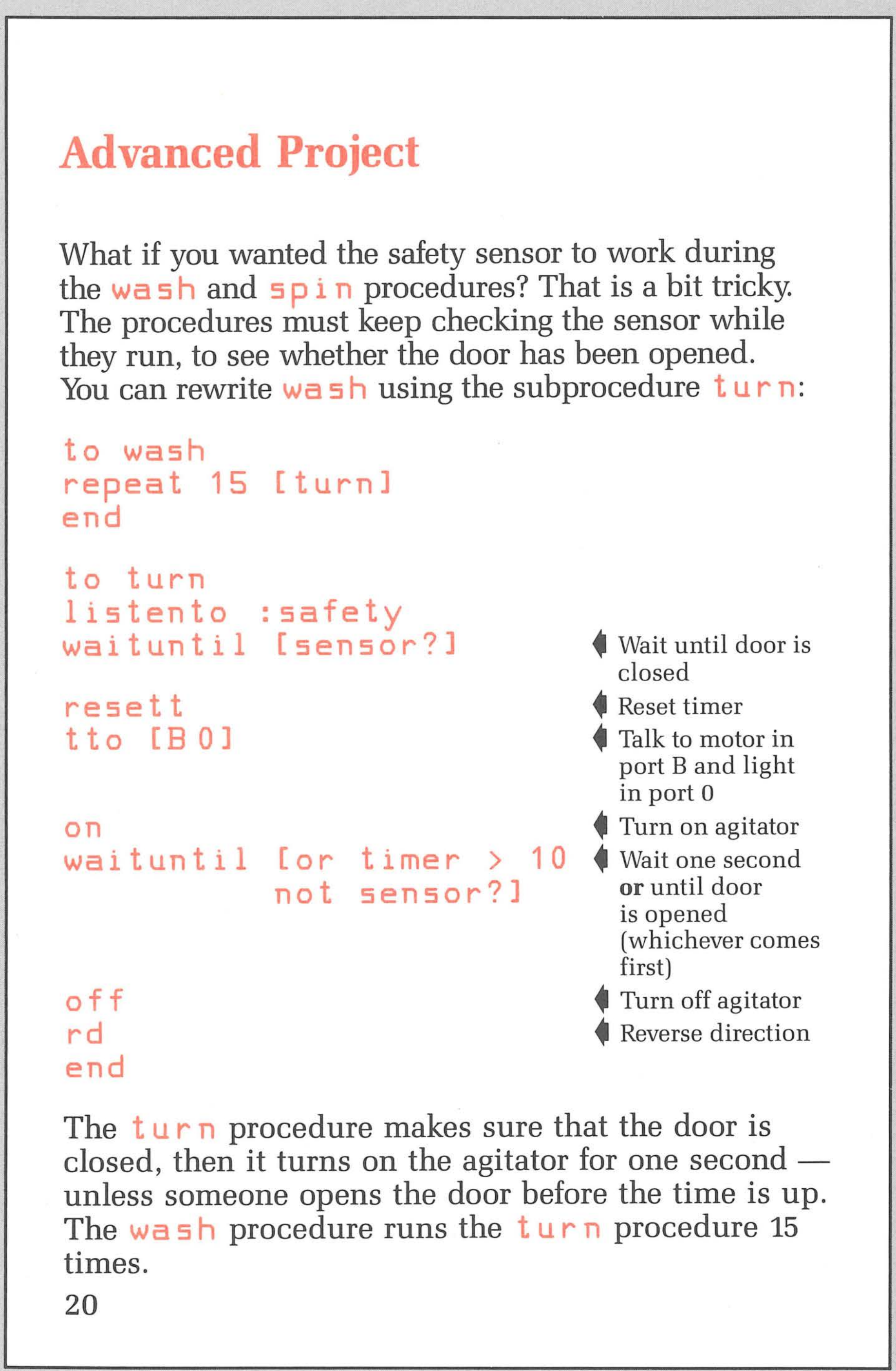
This procedure follows the suggestion. Obviously, there are many other possibilities.

```
to combination
listento :starter
waituntil [sensor?]          ◀ Wait until the sensor is pressed
listento :safety
waituntil [sensor?]          ◀ Wait until the door is closed
on
waituntil [not sensor?]      ◀ Wait until the door is opened
off
end
```

When you open the door, you only interrupt the `wash` procedure, you don't end it. When you close the door, the machine will run the remaining part of the cycle.

Students might try to write a new version of `spin` that stops whenever someone opens the door.

Boolean logic. The "advanced projects" introduce the use of the LEGO® TC logo primitive `or`. The connectives "or" and "and" are very important in logic and computer science — as well as in everyday English.



Advanced Project

What if you wanted the safety sensor to work during the `wash` and `spin` procedures? That is a bit tricky. The procedures must keep checking the sensor while they run, to see whether the door has been opened. You can rewrite `wash` using the subprocedure `turn`:

```
to wash
repeat 15 [turn]
end

to turn
listento :safety
waituntil [sensor?]                  ◀ Wait until door is
                                       closed
resett                               ◀ Reset timer
tto [B 0]                            ◀ Talk to motor in
                                       port B and light
                                       in port 0
on                                   ◀ Turn on agitator
waituntil [or timer > 10             ◀ Wait one second
              not sensor?]             or until door
                                       is opened
                                       (whichever comes
                                       first)
off                                  ◀ Turn off agitator
rd                                   ◀ Reverse direction
end
```

The `turn` procedure makes sure that the door is closed, then it turns on the agitator for one second — unless someone opens the door before the time is up. The `wash` procedure runs the `turn` procedure 15 times.

You can build and program many other household machines. Here are some ideas:

21

The Automated House. Students can design, build, and program other machines for an "automated house." Some possibilities: sewing machines, toasters, automatic doors.

The Home of the Future. What will homes look like 100 years from now? Students can draw pictures and write essays about their ideas. They might also look at old books and magazines to find out about previous predictions. Many people in the 1940's and 1950's made predictions about what homes would be like in the 1980's. How accurate were they?

Project 3: Conveyor Belt

Key Ideas

* Project themes: Automation, Factory of the Future
* Social studies: Industrial Revolution
* Science: Intelligent machines, robots

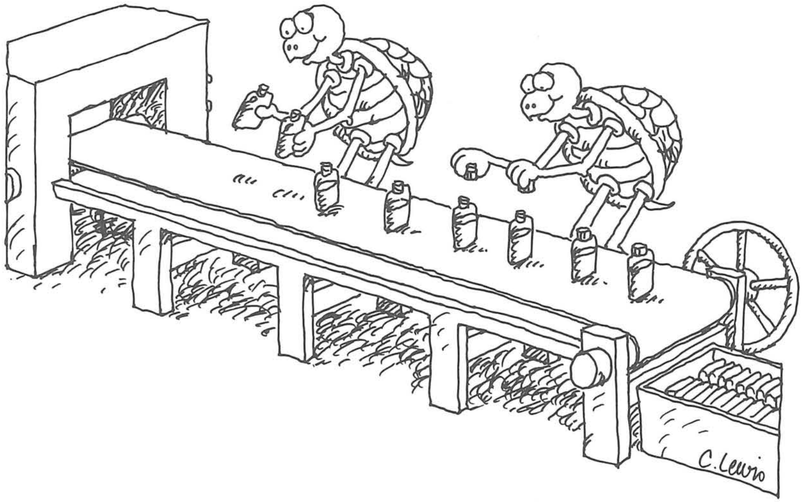
Industrial Revolution. This is a good opportunity to discuss the history of industrialization. You can discuss: important inventors of the Industrial Revolution (Edison, Fulton, etc.); important machines in the Industrial Revolution (light bulb, steam engine); changing the nature of production (one-at-a-time custom work vs. mass production); changing the nature of work (craftspeople vs. specialists along assembly line); broader societal changes (urbanization, pollution, more consumer goods).

Factory automation. Many factories are in the process of change. They are installing robots and other computerized systems to do some of the work that people used to do. This automation of the factory has many advantages. Robots can do dangerous and unpleasant jobs that people don't want to do. And factories using robots can sometimes make products more cheaply than before (robots can work 24 hours a day and they don't take coffee breaks).

But there are problems too. As automation eliminates jobs in the factory, where will people find work? Automation is also creating many new jobs (building robots, programming computers). But will the people who lose their jobs in the factory have the right skills for these new jobs? What if the new jobs are in a different part of the country?

Project 3

Machines at Work: Conveyor Belt



Companies use machines for two major reasons:

*By using machines, companies can make some products more **efficiently**. That is, they can make the same number of products for less money — or they can make more products for the same amount of money.

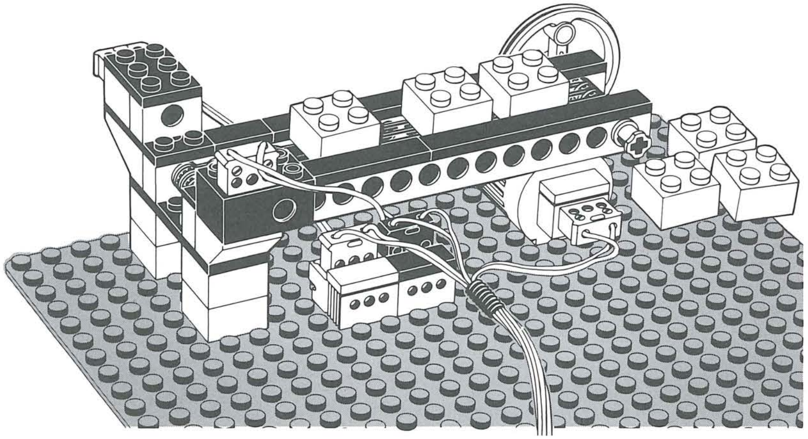
*By using machines, companies can make **new products** that would not have been possible without machines.

22

The conveyor belt is an important machine in many factories. Conveyor belts carry things from one place to another. When people work along a conveyor belt, it is called an **assembly line**.

Assembly lines have changed the way many people work. The conveyor belt carries products from one person to the next, and each person does a small part of the overall job. One person might add some nuts and bolts, while another person paints part of the product.

Activity Card 5 shows how to build a LEGO® conveyor belt.



23

Assembly lines have made factories more efficient. But many people think that assembly lines make work less enjoyable.

This is an example of **specialization**.

Factory of the Future. Students can design and build other machines for an automated factory, such as elevators, automatic doors, robot arms, and walking robots. They might also draw pictures of what they expect factories to look like in the future.

The Automated Classroom. Students can simulate a factory assembly line in the classroom. Some students might try to build a LEGO® model (like a washing machine or robot arm) along an assembly line, with each student doing a small part of the job. Other students might try to build the model by themselves. Is the assembly-line approach more efficient?

Counting Objects

You can use an optosensor to count the number of objects that move along the conveyor belt.

Imagine that the conveyor belt is carrying LEGO® bricks. When a brick passes in front of the sensor, it blocks the light. That increases the value of the sensor's counter. (The counter increases by one every time the sensor changes from `false` to `true`).

For a more complete description of the counter, see page 48 of *Getting Started* or the *Reference Guide*.

So the value of the counter will tell you how many bricks have moved past the optosensor. `dozen` will wait until 12 objects have moved past the optosensor, then it will turn off the belt.

```
to dozen
talkto "A    ◀ Talk to the motor
on
listento 6
resetc
waituntil [counter = 12]
off
end
```

Turn on the light, then run `dozen`. It is best to use **blue** LEGO® bricks, with the "studs" facing the optosensor.

Red bricks do not work well. Black bricks sometimes work, but blue bricks are best.

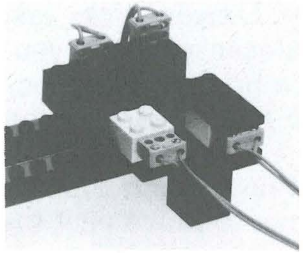
BUG BOX

The bricks aren't blocking all the light.
You aren't "listening to" the correct port.

Measuring Length

You can also use optosensors to tell the difference between long objects and short ones.

Let's say you want to detect all objects longer than 2.5 centimeters. Place two optosensors next to one another:



The "eyes" of the optosensors are then about 2.5 centimeters apart. So any object longer than 2.5 centimeters will block **both** optosensors at once.

`sort` will turn off the conveyor belt whenever the sensors "see" an object longer than 2.5 centimeters.

```
to sort
talkto "A                  ◀ Talk to the motor
on
listento [6 7]             ◀ Listen to both sensors
waituntil [sensor?         ◀ Wait until both
      = [true true]]         sensors blocked
off
end
```

Before using `sort`, turn on both lights: `talkto [2 3] on`. Be sure to use dark-colored objects (like blue LEGO® bricks).

The TC 0 building set contains only one optosensor. Have students share or purchase additional sensors through the LEGO® spare parts service.

Remember: When you are "listening to" both sensors, `sensor?` returns a list of **both** values. There are four possibilities: `[true true]`, `[true false]`, `[false true]`, `[false false]`.

You could change the `sort` procedure so that it turns on both lights. But this approach might not work, since the procedure might check `sensor?` before the lights have had time to fully "light up."

Advanced Project

You can also use optosensors to calculate the actual **length** of any LEGO® brick that moves along the conveyor belt. In fact, you need only **one** optosensor!

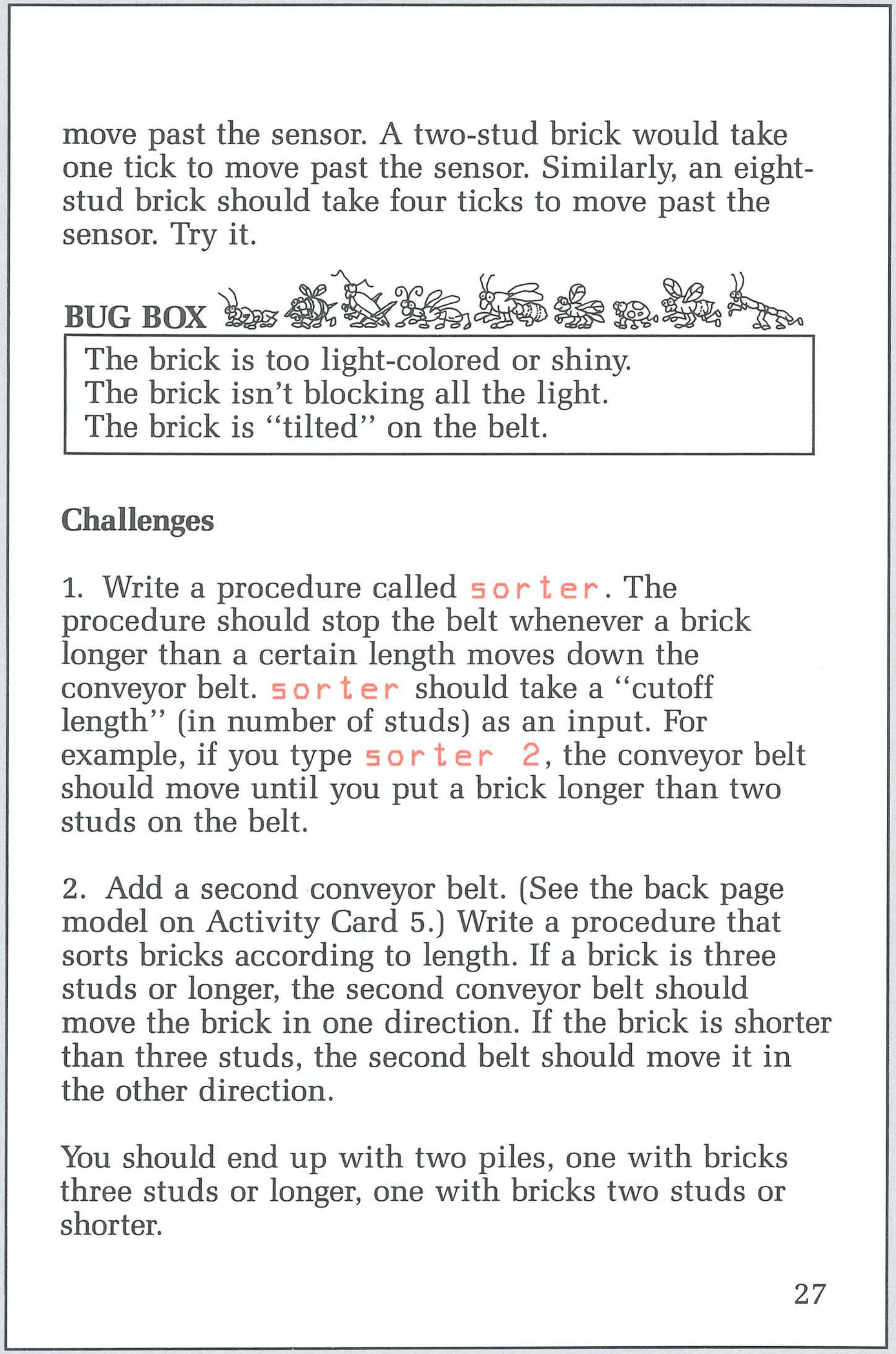
Here is one strategy: Longer bricks take a longer time to move past the optosensor. So if you measure how much time it takes a brick to move past the optosensor, you can figure out how long the brick is.

Start by finding out how much time it takes for a brick, four studs long, to move past the optosensor. You can use a procedure like this:

```
to measure
talkto "A                       ◀ Talk to the motor
on
listento 6
waituntil [sensor?]             ◀ Wait until brick
                                  blocks light
resett                          ◀ Reset the timer
waituntil [not sensor?]         ◀ Wait until light
                                  not blocked
show timer                      ◀ Print the time
off
end
```

Turn on the LEGO® light and run `measure`. Put a four-stud brick on the belt. Let's say that the computer prints `2`. So it took two ticks of the timer (that is two tenths of one second) for the brick to

It is a good idea to take several measurements, then take the average.



move past the sensor. A two-stud brick would take one tick to move past the sensor. Similarly, an eight-stud brick should take four ticks to move past the sensor. Try it.

BUG BOX

The brick is too light-colored or shiny.
The brick isn't blocking all the light.
The brick is "tilted" on the belt.

Challenges

1. Write a procedure called `sorter`. The procedure should stop the belt whenever a brick longer than a certain length moves down the conveyor belt. `sorter` should take a "cutoff length" (in number of studs) as an input. For example, if you type `sorter 2`, the conveyor belt should move until you put a brick longer than two studs on the belt.

2. Add a second conveyor belt. (See the back page model on Activity Card 5.) Write a procedure that sorts bricks according to length. If a brick is three studs or longer, the second conveyor belt should move the brick in one direction. If the brick is shorter than three studs, the second belt should move it in the other direction.

You should end up with two piles, one with bricks three studs or longer, one with bricks two studs or shorter.

27

Challenges

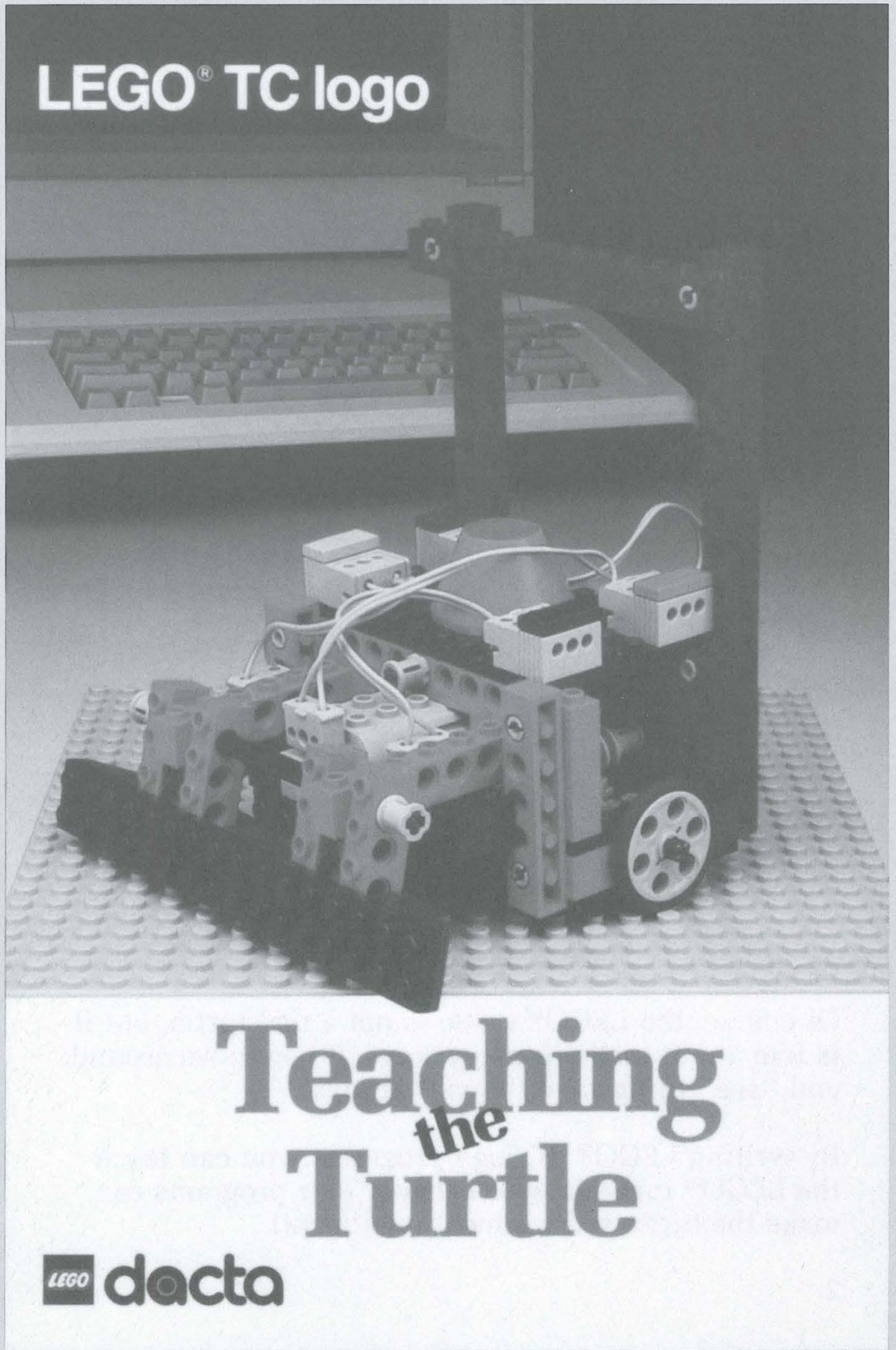
1. Assume that each stud takes one "tick" of the timer to move past the sensor:

```
to sorter :length
talkto "A
on
listento 6
waituntil [sensor?]
resett
waituntil [not sensor?]
if timer > :length [off stop]
sorter :length
end
```

In the `if` statement, `off` turns off the motor and light, and `stop` stops the procedure. Make sure the light is on before using `sorter`.

2. Here is one approach. It uses the LEGO® TC logo primitives `seteven` and `setodd`. These primitives set the **direction** that a motor should turn (either the **even** direction or the **odd** direction). See the *Reference Guide* for more details. Make sure the light is on before running `twobelts`.

```
to twobelts
talkto "A
on                                   ◀ Turn on the motor for first
                                       belt
listento 6
waituntil [sensor?]                  ◀ Wait until "see" object
resett                               ◀ Start the timer
waituntil [not sensor?]              ◀ Wait until the object is
                                       past
ifelse timer < 3 [short][long]       ◀ Sort according to time
twobelts                             ◀ Start whole process again
end

to long
talkto "B                            ◀ Talk to motor for second
                                       belt
seteven                              ◀ Set to the "even" direction
onfor 20
end

to short
talkto "B                            ◀ Talk to motor for second
                                       belt
setodd                               ◀ Set to the "odd" direction
onfor 20
end
```



Chapter 8
Teacher's Notes for *Teaching the Turtle*

The *Teaching the Turtle* booklet focuses on the **programming** side of LEGO® TC logo. Students program the LEGO® turtle to "behave" in various ways. These activities provide a natural framework for discussing animal behavior — and the abilities of computers to mimic animal behavior.

The booklet can be used in two very different ways:

* **The Turtle as an Introductory Activity.** Over the past 25 years, the turtle has proven a very effective device for introducing children to programming. You might want to build the LEGO® turtle yourself and use it as a first introduction to programming (even before the *Getting Started* booklet). This approach can be particularly effective with young children (under the age of nine).

There is one problem with this approach: the *Teaching the Turtle* booklet does not provide a full introduction to the interface box or to programming. For example, it does not explain primitives like `repeat` or the use of LEGO® TC logo procedures. To learn these concepts, you or your students would have to refer to the *Getting Started* booklet.

The first half of this chapter (pages 114-123) covers these introductory activities.

* **The Turtle as an Advanced Activity.** The LEGO® turtle can also be used for advanced programming projects. These projects become possible after students add sensors to the turtle. Sensors enable the turtle to get information from its environment (much as a real animal would). Students can then write programs that teach the turtle how to respond to various "stimuli" in the environment.

Advanced students should still begin with the introductory turtle activities (to become familiar with the special turtle commands). But these students can move through the introductory activities rather quickly.

The second half of the teacher's notes (pages 124-141) covers advanced turtle-programming activities.

Introductory Activities

Key Ideas

* Setting up the LEGO® turtle
* Introductory turtle programming
* Animals vs. machines
* Math concepts: Measurement, angles, ratios
* New primitives: `tfd`, `tbk`, `trt`, `tlt`

Young students (under age ten) will probably need help in building the turtle.

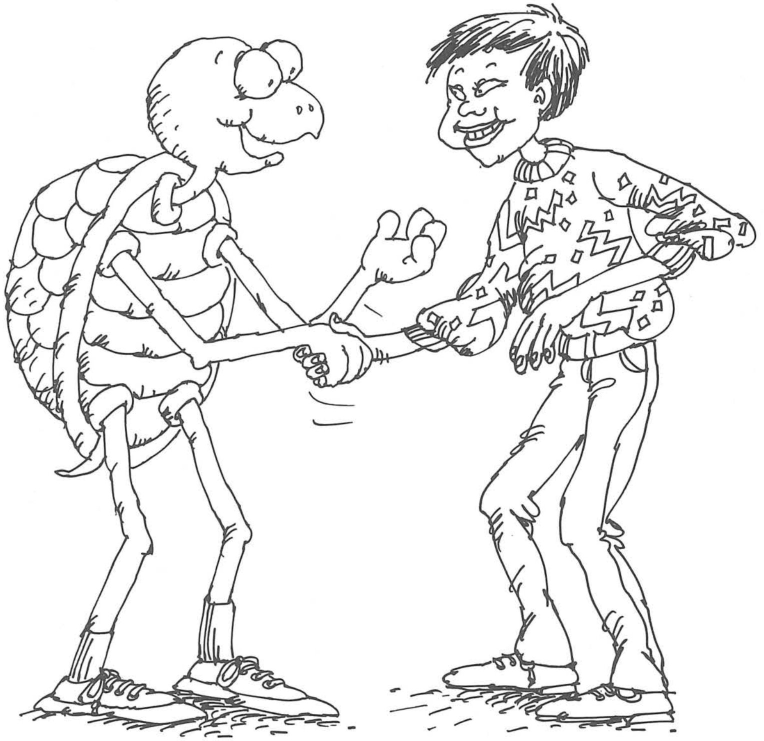
History of "Turtles." Imitation turtles have a long tradition, both in education and in the field of machine intelligence. An English researcher named Grey Walter built a mechanical turtle in the early 1950's and used it to study intelligence and animal behavior. Seymour Papert made the turtle a central object in the computer language Logo. In early work with Logo, children gave commands to a "floor turtle," which was similar to the LEGO® turtle. Later, when graphic computer screens became available, Papert added the Logo screen turtle.

Machines vs. Animals. It is worthwhile to discuss the similarities and differences between animals and machines. You might ask questions such as: How are animals like machines? How are they different? What types of decisions do animals have to make? What information do they need to make those decisions? Could machines make those decisions?

This type of discussion can give students a better appreciation for the capabilities of both machines **and** animals. By using the LEGO® turtle, students will see that computers and machines can perform some pretty impressive tasks. But at the same time, these activities show that animals are superior in many ways. There are many tasks (such as telling the difference between a dog and a cat) that most animals find easy, but even today's most sophisticated computers find very difficult.

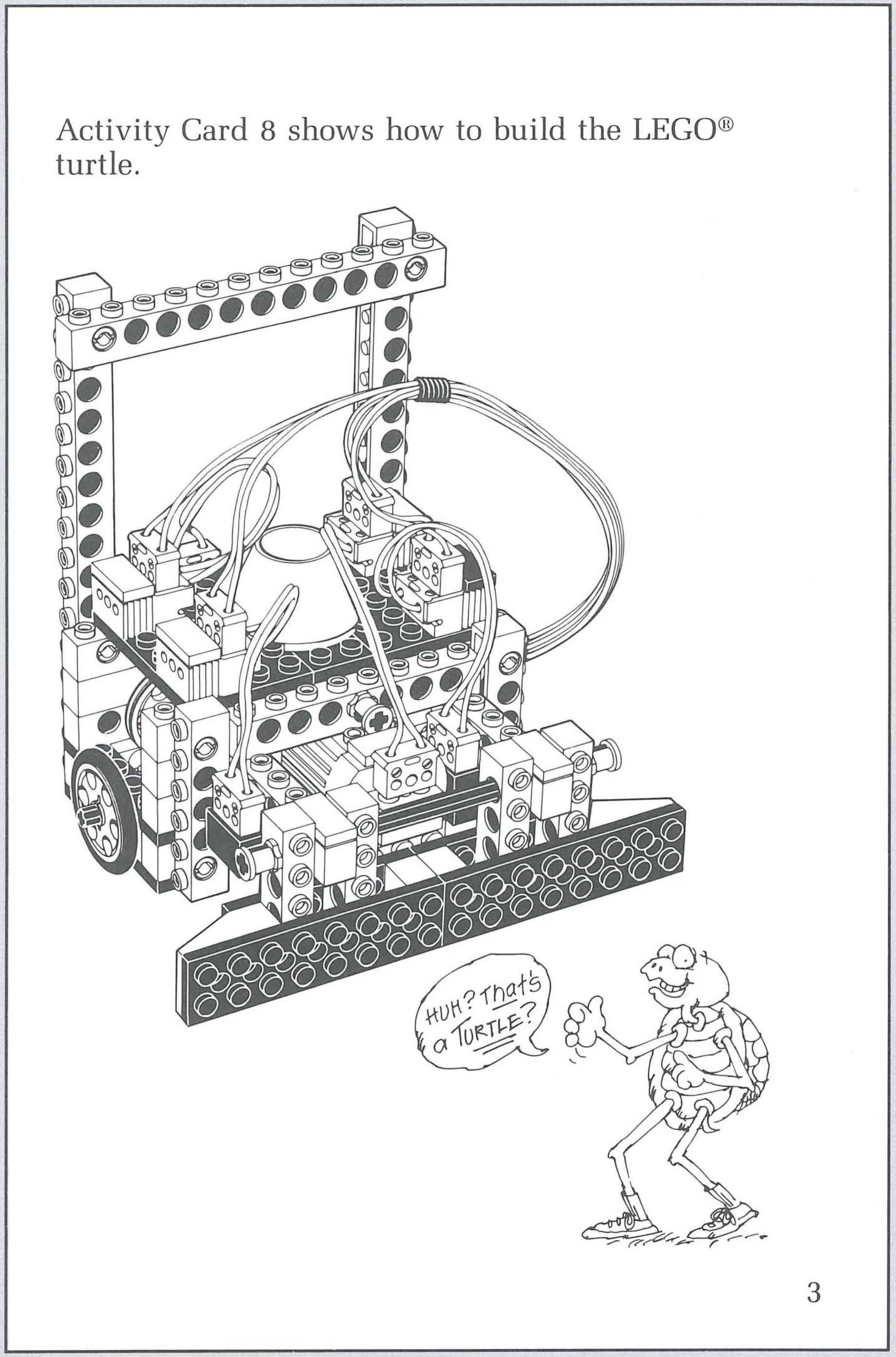
Meet the Turtle

You are about to meet the LEGO® turtle!



Of course, the LEGO® turtle is not a real turtle, but it is like a real turtle in many ways. It can move around and "see" things and "touch" things.

By writing LEGO® TC logo programs, you can teach the LEGO® turtle how to behave. Your programs can make the turtle seem almost intelligent!



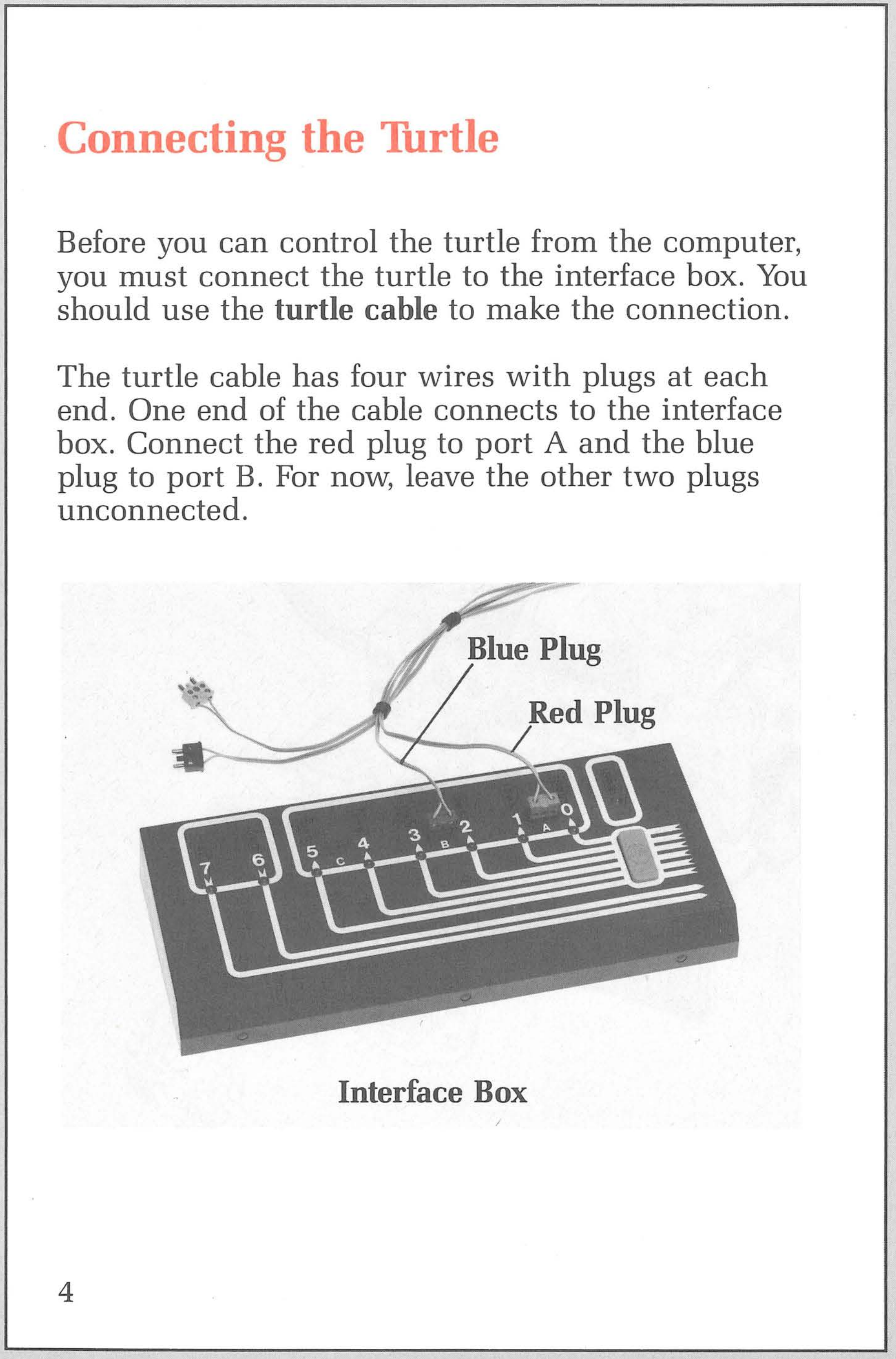
Activity Card 8 shows how to build the LEGO® turtle.

3

Mobile robots. Most "robots" used in factories look like mechanical arms. The LEGO® turtle is quite different. For one thing, it can move around. In this way, the turtle is much more like the robots that children have seen in science-fiction movies like *Star Wars* (although the LEGO® turtle is not nearly as "intelligent"). You might discuss how mobile robots could be used in the real world. Some possibilities: delivering the mail, entering burning buildings to fight fires, guarding buildings at night.

Playing turtle. By "playing turtle" (acting out what the turtle should do), students can gain a better understanding of programming. Students could write procedures for other students to act out.

Robots. Students could build the robot arm on the back of Activity Card 5. Also, students could "invent" their own robots. First, they could draw pictures of robots in their Inventor's Notebooks. Then they could write stories about the robots, or actually build them out of LEGO® bricks. Students could build on top of the LEGO® turtle itself or build their own original robots. To get ideas, students could look through books and magazines for pictures and stories about robots.



Connecting the Turtle

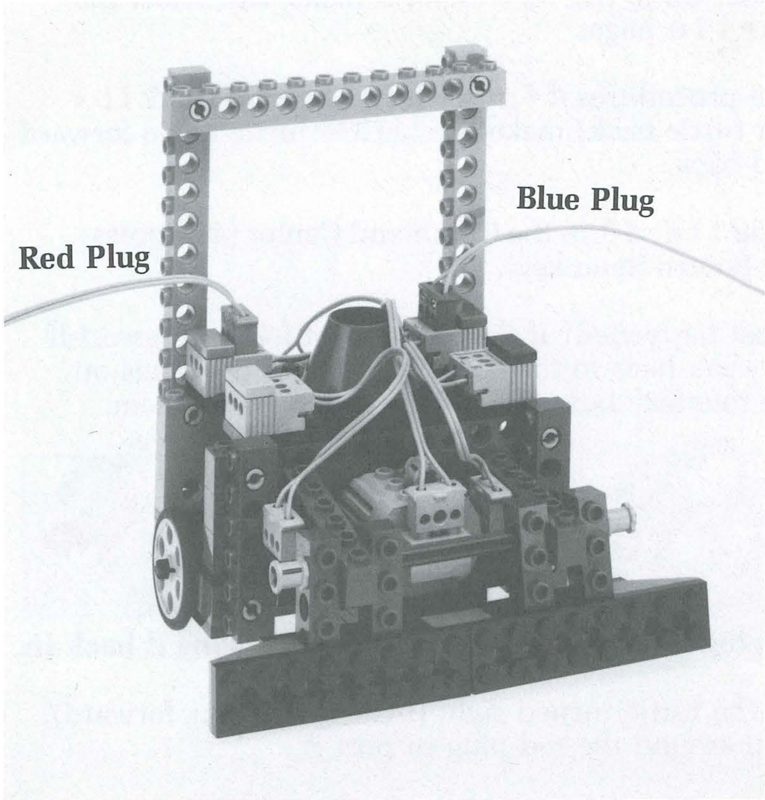
Before you can control the turtle from the computer, you must connect the turtle to the interface box. You should use the **turtle cable** to make the connection.

The turtle cable has four wires with plugs at each end. One end of the cable connects to the interface box. Connect the red plug to port A and the blue plug to port B. For now, leave the other two plugs unconnected.

Interface Box

The other plugs will be used later for connecting sensors.

Connect the other end of the cable to the outlets on top of the turtle. Connect the red plug to the outlet with the red plate. Connect the blue plug to the outlet with the blue plate.



The blue outlet connects to the front motor, which controls the left wheel. So the wire with the blue plug controls the left wheel. Similarly, the wire with the red plug controls the right wheel.

5

It is useful to trace through all of these connections. First trace how the outlets connect to the motors. Then pick up the turtle and look underneath to see how each motor connects to one of the wheels. The motor drives a rubber band which drives a pulley which drives a worm gear which drives a 16-tooth gear which drives the wheel. Whew! The gears and pulleys "gear down" the turtle so that it moves more slowly, and with more power. (See pages 40-43 in *Getting Started* for further explanation of gears.)

Apple users: Remember to use the LEGO® TC logo Operating Disk.

IBM users: Remember to create a disk on which you can store your own pages. See the *Setup Guide* for more information.

The `Contents` menu appears when you first load the software. To return to `Contents` menu at any later time, simply press the Escape (Esc) key.

You have to go through this process of adjusting the plugs every time you re-connect the turtle.

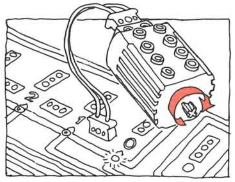
Turtle Commands

The **turtle** page on the LEGO® TC logo disk has special procedures that let you control the LEGO® turtle. Go to the `Contents` menu and select the `turtle` page.

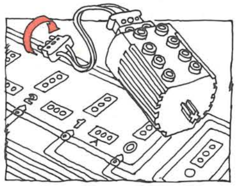
The procedures `tfd` (for **t**urtle **f**orwar**d**) and `tbk` (for **t**urtle **b**ac**k**) make the LEGO® turtle move forward and back.

Type `tfd 10` in the Command Center (and press the Return Enter key).

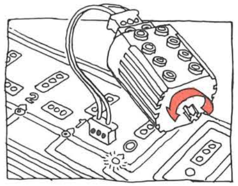
What happened? If the turtle went forward, great! If not, you have to turn around some of the plugs on the interface box. Here's how to fix the problem:



Unplug it



Flip it!

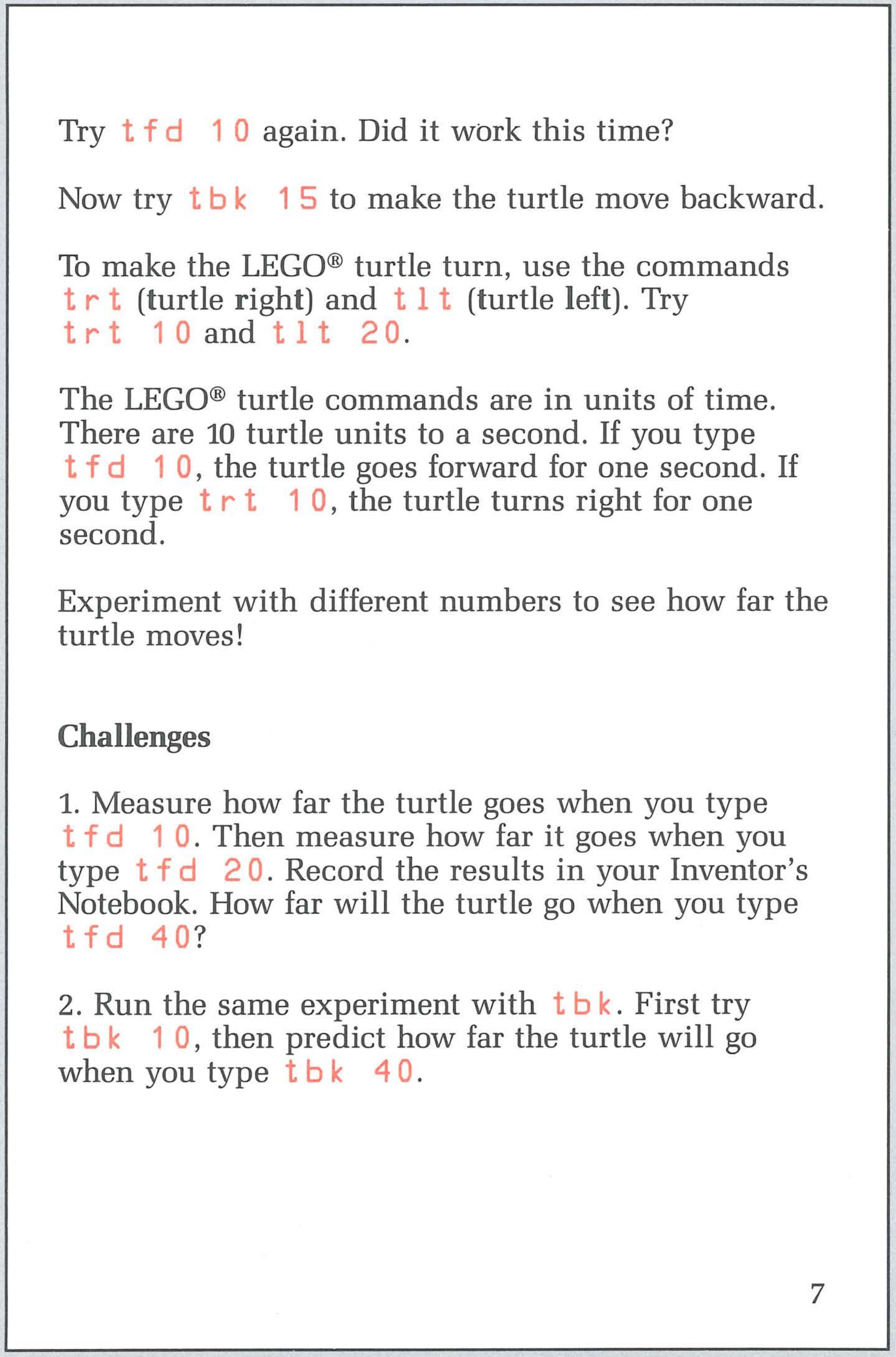


Plug it back in

*If the turtle turned right (instead of going forward), turn around the red plug in port A.

*If the turtle turned left, turn around the blue plug in port B.

*If the turtle went backwards, turn around the plugs in **both** port A and port B.



Try `tfd 10` again. Did it work this time?

Now try `tbk 15` to make the turtle move backward.

To make the LEGO® turtle turn, use the commands `trt` (turtle **r**igh**t**) and `tlt` (turtle **l**ef**t**). Try `trt 10` and `tlt 20`.

The LEGO® turtle commands are in units of time. There are 10 turtle units to a second. If you type `tfd 10`, the turtle goes forward for one second. If you type `trt 10`, the turtle turns right for one second.

Experiment with different numbers to see how far the turtle moves!

Challenges

1. Measure how far the turtle goes when you type `tfd 10`. Then measure how far it goes when you type `tfd 20`. Record the results in your Inventor's Notebook. How far will the turtle go when you type `tfd 40`?

2. Run the same experiment with `tbk`. First try `tbk 10`, then predict how far the turtle will go when you type `tbk 40`.

7

Notice that the units are different than those for the LEGO® TC logo screen turtle. If you type `trt 90`, the LEGO® turtle will turn right for **9 seconds**, not for 90 degrees. You can recalibrate the turtle commands so that they take inputs in degrees. See pages 27 and 28 in *Teaching the Turtle*.

Motion and time. When students think about how the LEGO® turtle moves, they think about distance (`tfd`, `tbk`) and angles (`trt`, `tlt`). But inputs to the turtle are in time. The turtle, in effect, is converting time into distance and angles. The conversion is based on the **speed** of the turtle. Since the turtle moves at a **constant speed**, a given number of seconds will always convert to the same distance or angle. This is based on the formula:

Distance = speed (a constant) * time

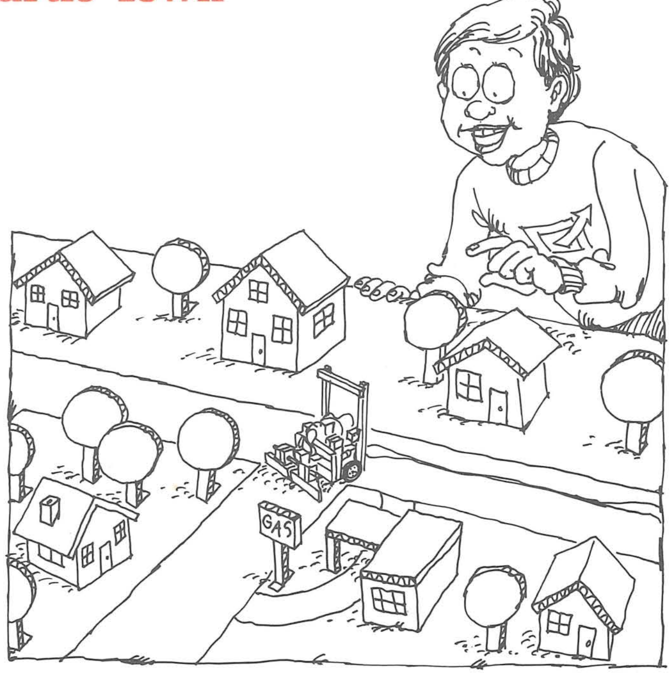
Challenges

1. If the turtle goes 6 centimeters when you type `tfd 10`, then it should go 12 centimeters when you type `tfd 20`, and it should go 24 centimeters when you type `tfd 40`.

Note: The turtle might not move in a straight line. There are two possible reasons: (1) There is more friction on one wheel/axle than the other, (2) One motor is "stronger" than the other. You can reduce the first problem by making sure none of these pieces are rubbing against one another.

2. The turtle will go about the same distances for `tbk 10` and `tfd 10`.

Turtle Town



You can set up a "turtle town" in your classroom. Draw a plan for the town on a large piece of paper. Show the streets and places for houses, stores, and playgrounds. You can use LEGO® bricks or other materials (like milk cartons) to make buildings for your town.

Steer the turtle through the town by typing these commands in the Command Center: `tfd`, `tbk`, `trt`, and `tlt`. Make sure the streets are wide enough for the turtle!

You can write procedures that tell the turtle how to get to different places in the town. (See page 20 of the *Getting Started* book to learn more about procedures.)

Important note: There are already some procedures on the `turtle` page. You need these procedures to make the turtle work. Do not erase them.

When you add your own procedures, you should give a new name to the `turtle` page. That way, the original `turtle` page will not be changed (and you can use it again later).

Give the page a new name like `town`.
Type `namepage "town`
(or `np "town`) in the Command Center.

Now, you can write a procedure that moves the turtle from its home to the school. The procedure will depend on the layout of your turtle town. Here's one possibility:

```
to home.to.school
tfd 16
trt 12
tfd 27
end
```

If you are using the LEGO® turtle as an introduction to programming, you can introduce procedures at this point.

There are nine procedures already on the `turtle` page. For a more complete explanation of how these procedures work, see pages 22-25 in *Teaching the Turtle.*

Note that you can use characters other than letters (like underlines and periods) in procedure names.

Make sure the pen fits through the **center** hole underneath the turtle.

You might want to have the students "play turtle." That is, the students can make believe that they are turtles, and act out the turtle commands themselves. For example, students acting out the line

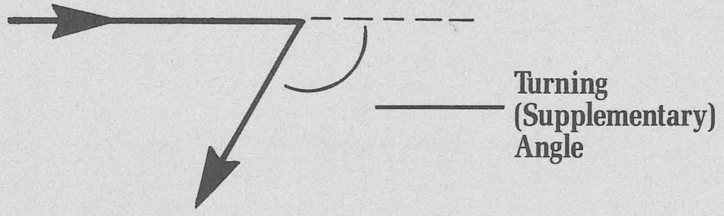
```
repeat 3 [tfd 10 tbk 10]
```

would walk back and forth three times. This type of experience gives students a new, more personal way to think about programming.

Challenges (Referring to page 11 of *Teaching the Turtle*.)

1. The command `trt 10` should make the turtle turn through twice as large an angle as `trt 5`.

Be careful: The angle that is drawn on the paper is **not** the angle you want to measure. The supplementary angle is the one you want to measure.



2. Younger students can do this by estimation or trial-and-error. To be more systematic, first measure the angle the turtle turns for `trt 10`. If it turns 80 degrees, you should solve the ratio:

$$\frac{\text{trt 10}}{80} = \frac{\text{trt ??}}{90}$$

In this case, **??** equals **11.25**. So `trt 11` will make the turtle turn about 90 degrees. Note: It is difficult to make the turtle movements precise because `onfor` truncates the decimal from its input value. Students should try to get within a couple of degrees.

Drawing with the Turtle

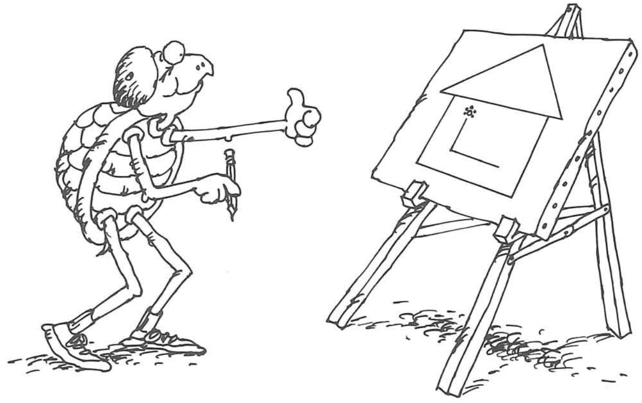
You can draw shapes and pictures with the LEGO® turtle. Just place the turtle on top of a piece of paper, and put a soft-tipped marking pen through the hole in the turtle. Make sure the pen touches the paper.

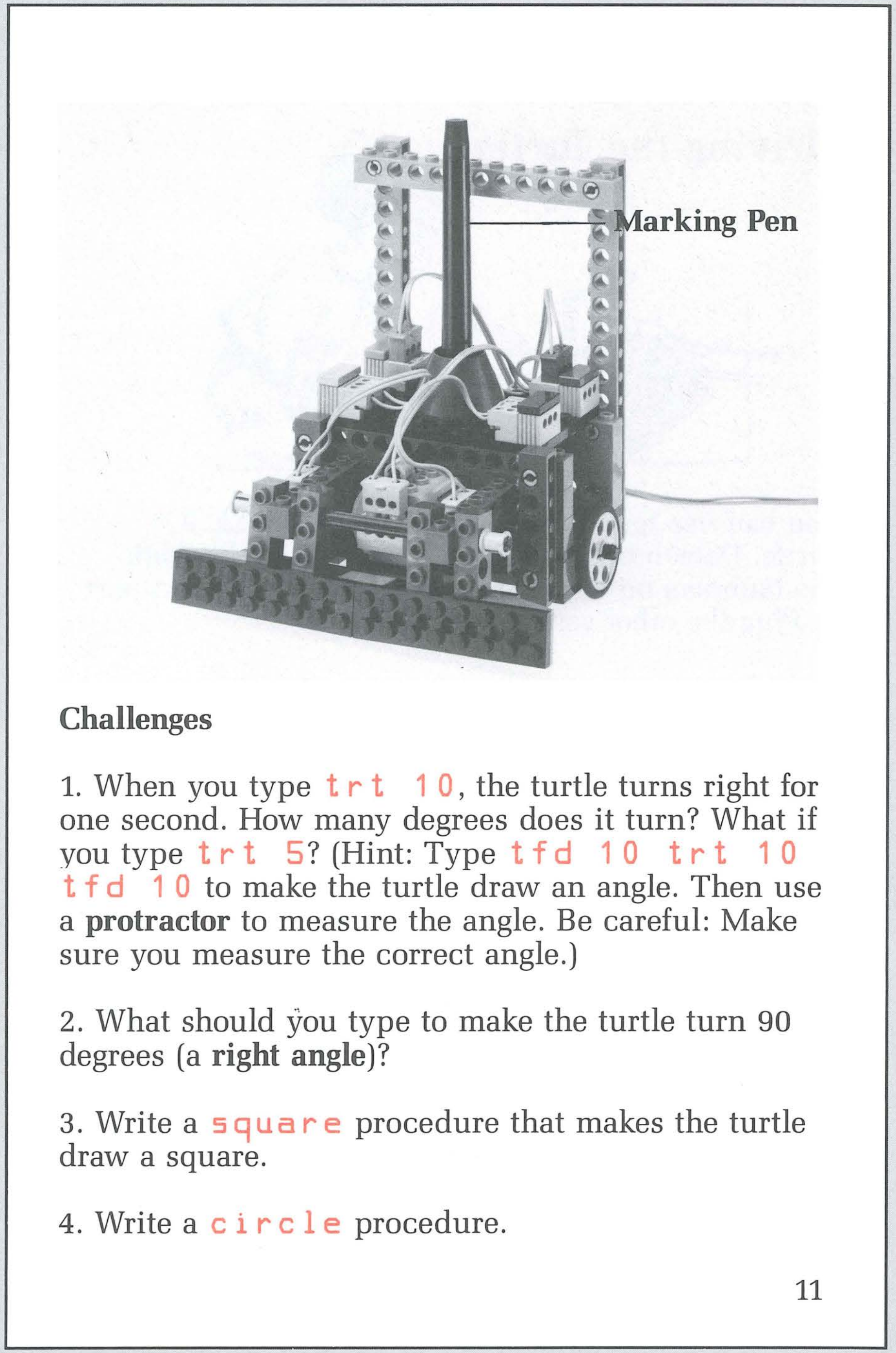
Experiment with `tfd`, `tbk`, `trt`, `tlt`.

For example, try typing each of these in the Command Center.

```
tfd 10 trt 5 tfd 15

repeat 3 [tfd 10 trt 12]

repeat 4 [tfd 10 tlt 5]
```

Remember that the inputs to `trt` and `tlt` are in units of time (ticks), not degrees. (See page 18 of *Getting Started* to learn more about `repeat`.)




Marking Pen

Challenges

1. When you type `trt 10`, the turtle turns right for one second. How many degrees does it turn? What if you type `trt 5`? (Hint: Type `tfd 10 trt 10 tfd 10` to make the turtle draw an angle. Then use a **protractor** to measure the angle. Be careful: Make sure you measure the correct angle.)

2. What should you type to make the turtle turn 90 degrees (a **right angle**)?

3. Write a `square` procedure that makes the turtle draw a square.

4. Write a `circle` procedure.

11

3. If `trt 11` makes the turtle turn a right angle, then the `square` procedure should be something like this:

```
to square
repeat 4 [tfd 20 trt 11]
end
```

Of course, changing the input to `tfd` will change the size of the square.

4. The `circle` procedure should look something like this:

```
to circle
repeat 60 [tfd 1 trt 1]
end
```

You will probably need to change the input `60` if you don't want the turtle to retrace part of the circle. To do this systematically, first measure how far `trt 1` makes the turtle turn. If it turns eight degrees, then you should need:

$$\frac{\text{360 degrees in a circle}}{\text{8 degrees per turn}} = \text{45 turns}$$

In this case you would use `repeat 45 [tfd 1 trt 1]`.

You cannot use numbers less than 1 with the turtle commands because decimal values are truncated: `tfd .9` is the same as `tfd 0`.

Geometry and ratios. By working with the LEGO® turtle, students naturally deal with ideas involving geometry and ratios. The challenges in which students write procedures for `square` and `circle` touch on many of these ideas. Students might also try to program the turtle to draw other polygons. Most introductory books on Logo include numerous examples (see Bibliography).

Advanced Activities

These activities build upon skills and concepts that are introduced in the *Getting Started* booklet.

Key Ideas

- Using sensors with the turtle
- "Intelligent" behavior
- Simulations
- New procedures: `gofd`, `gobk`, `gort`, `golt`, `ot`, `offturtle`, `seteven`, `setodd`

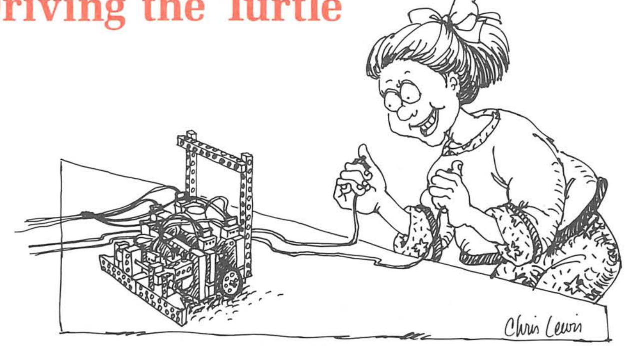
Remember: You must load the `turtle` page each time you want to use the turtle commands.

Intelligence. As the students write more sophisticated programs for the LEGO® turtle, you might discuss the idea of "machine intelligence." You might ask students: Do you think the LEGO® turtle is intelligent? What is intelligence? How could you make the turtle more intelligent? Could it ever be as intelligent as a real turtle? Could anyone ever program a computer to be as intelligent as a person? If so, would that be a good thing? What problems might it cause?

Real animals. Students might bring real animals into the classroom and study their behavior. Some possibilities: hampsters, ants (in an ant farm!), spiders, real turtles!

If you press both sensors, the turtle will move forward a little, then turn right a little, then move forward a little, then turn right a little, and so on. The turtle will end up turning in a circular arc.

Driving the Turtle



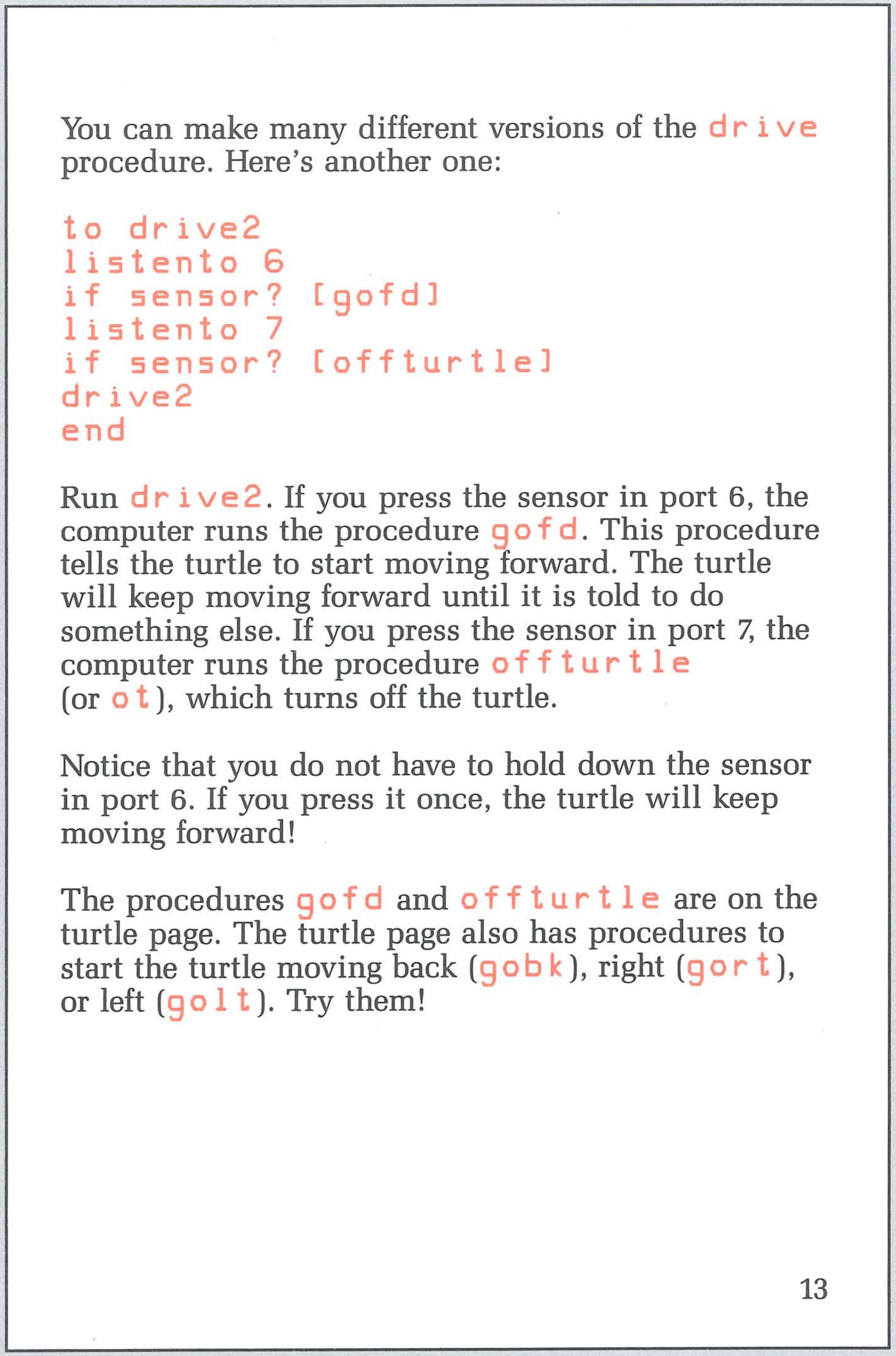
You can use touch sensors to "drive" the LEGO® turtle. Detach both touch sensors from underneath the bumpers on your turtle. Plug one sensor into port 6. Plug the other sensor into port 7.

```
to drive
listento 6
if sensor? [tfd 1]
listento 7
if sensor? [trt 1]
drive
end
```

Try `drive`. When you press the sensor in port 6, the turtle moves forward a little bit. When you press the sensor in port 7, the turtle turns right a little bit.

What happens when you press both sensors?

Remember: To stop a recursive procedure, hold down the Ctrl key and press the **S** key (**S** for **S**top). If the motors are still going, type `alloff`.



You can make many different versions of the `drive` procedure. Here's another one:

```
to drive2
listento 6
if sensor? [gofd]
listento 7
if sensor? [offturtle]
drive2
end
```

Run `drive2`. If you press the sensor in port 6, the computer runs the procedure `gofd`. This procedure tells the turtle to start moving forward. The turtle will keep moving forward until it is told to do something else. If you press the sensor in port 7, the computer runs the procedure `offturtle` (or `ot`), which turns off the turtle.

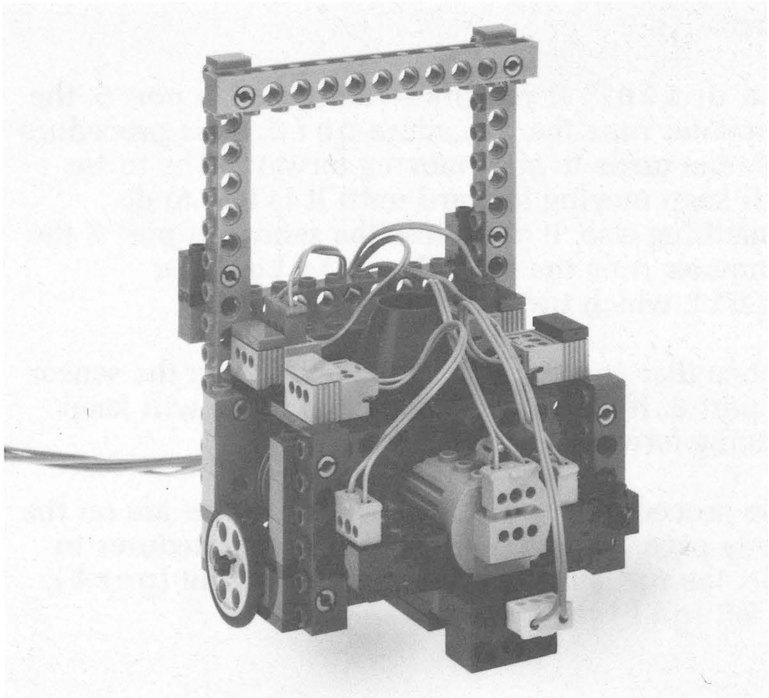
Notice that you do not have to hold down the sensor in port 6. If you press it once, the turtle will keep moving forward!

The procedures `gofd` and `offturtle` are on the turtle page. The turtle page also has procedures to start the turtle moving back (`gobk`), right (`gort`), or left (`golt`). Try them!

The relation between `tfd` and `gofd` is like the relation between `onfor` and `on`. Both `tfd` and `onfor` turn motors on for a specific amount of time. `gofd` and `on` turn motors on and leave them on (until they are told to do something else).

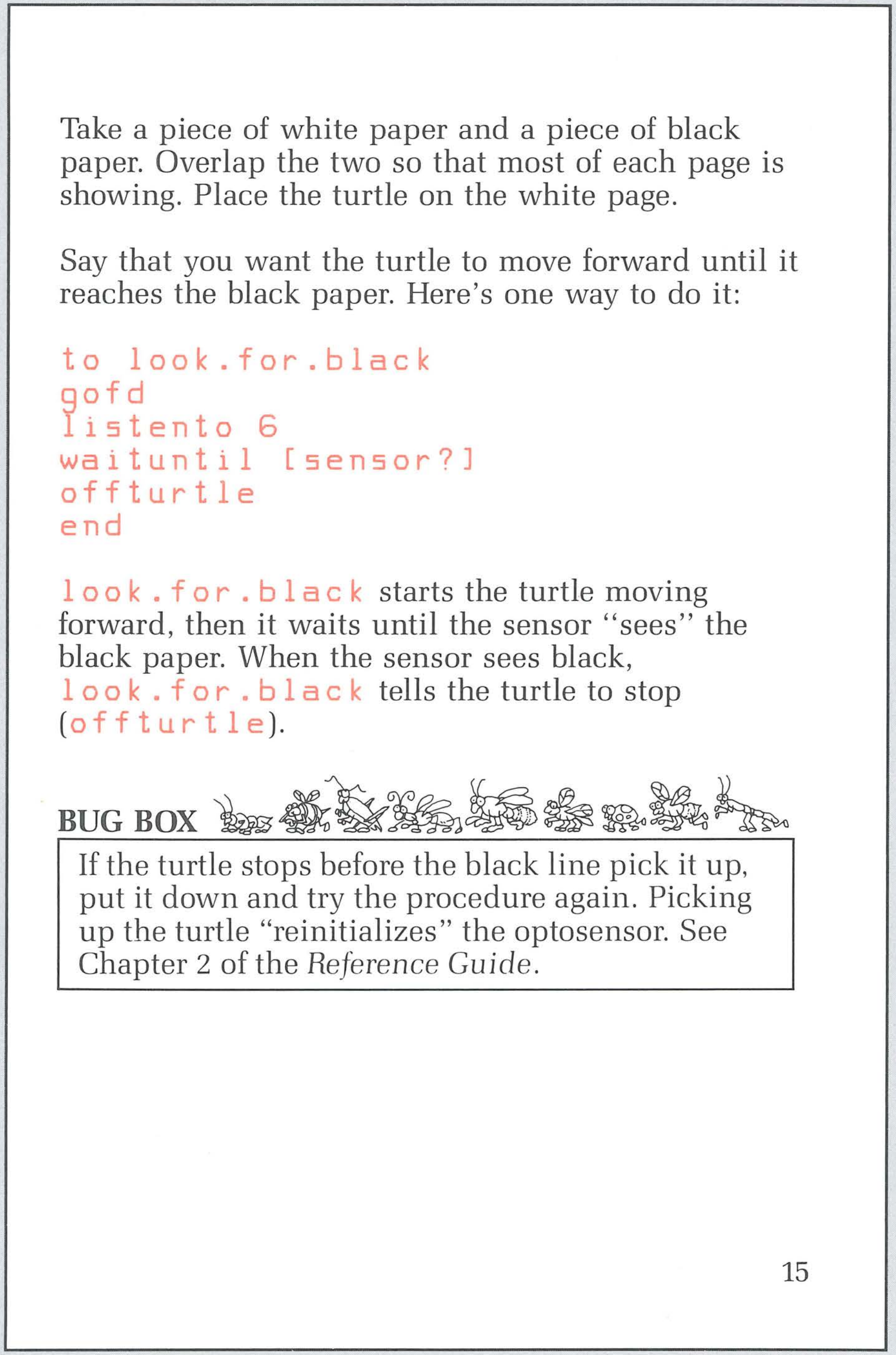
Seeing-Eye Turtle

If you add an optosensor to the LEGO® turtle, the turtle will be able to "see"! Of course, the turtle's "eyesight" is not very good. But it can tell the difference between black surfaces and white surfaces.



To start, connect the optosensor to the turtle, as shown on Activity Card 8. Then attach one end of the wire with the black plug into the black outlet on top of the turtle. Attach the other end into port 6.

The sensor should be about 30 millimeters (or about one-eighth of an inch) from the surface.



Take a piece of white paper and a piece of black paper. Overlap the two so that most of each page is showing. Place the turtle on the white page.

Say that you want the turtle to move forward until it reaches the black paper. Here's one way to do it:

```
to look.for.black
gofd
listento 6
waituntil [sensor?]
offturtle
end
```

`look.for.black` starts the turtle moving forward, then it waits until the sensor "sees" the black paper. When the sensor sees black, `look.for.black` tells the turtle to stop (`offturtle`).

BUG BOX

If the turtle stops before the black line pick it up, put it down and try the procedure again. Picking up the turtle "reinitializes" the optosensor. See Chapter 2 of the *Reference Guide*.

You can also use black tape on white paper.

Here's a different way to look for a black surface:

```
to look.for.black.2
listento 6
tfd 1
if sensor? [offturtle stop]
look.for.black.2
end
```

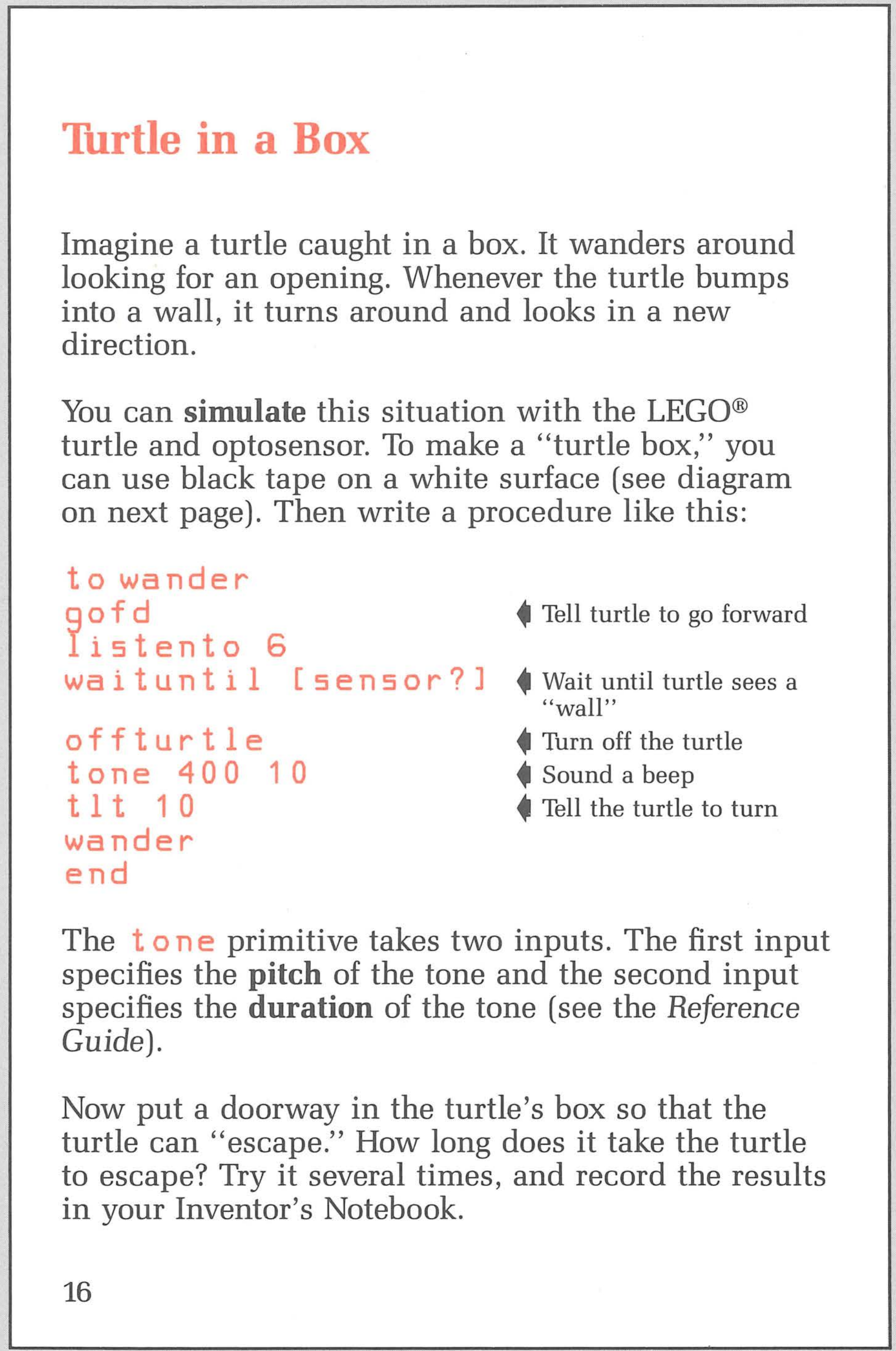
In this version, the turtle moves forward a little bit, then checks for the black. If it doesn't see black, it moves forward a little bit more, then checks for black again. And so on, until it sees black. The turtle's movement is "jerky" using this approach.

Notice that you need both `offturtle` (or `ot`) which turns off the turtle and `stop` (which stops the procedure).

The time the turtle takes to escape can vary significantly, depending on where you start the turtle. You can use this to demonstrate the importance of running an experiment several times.

Occasionally, the optosensor misreads the level of light. The longer the program runs, the greater the chance that the optosensor will give a faulty reading.

If a problem occurs stop the program (-S Ctrl-S) and turn off the motors and lights (`alloff`). Pick up the turtle and put it down again in the "box." Then rerun your program. Picking up the turtle exposes the optosensor to a sharp change in light, reinitializing the optosensor. For further explanation, see Chapter 2 of the *Reference Guide*.



Turtle in a Box

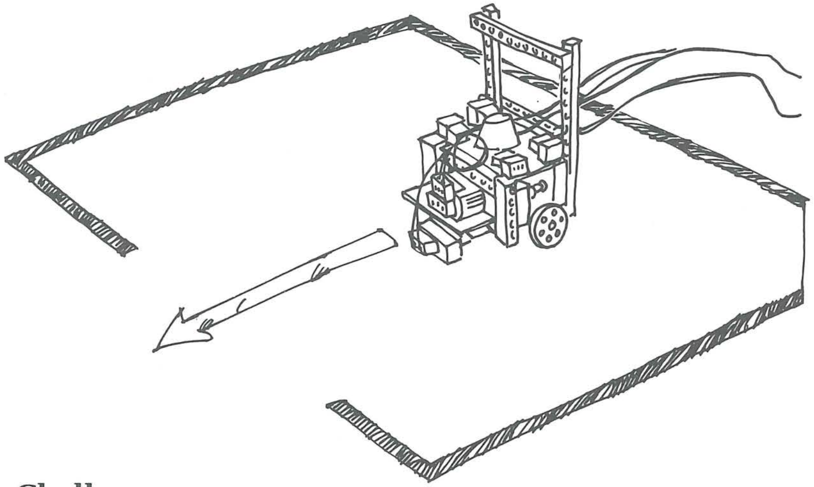
Imagine a turtle caught in a box. It wanders around looking for an opening. Whenever the turtle bumps into a wall, it turns around and looks in a new direction.

You can **simulate** this situation with the LEGO® turtle and optosensor. To make a "turtle box," you can use black tape on a white surface (see diagram on next page). Then write a procedure like this:

```
to wander
gofd                     ◀ Tell turtle to go forward
listento 6
waituntil [sensor?]      ◀ Wait until turtle sees a
                           "wall"
offturtle                ◀ Turn off the turtle
tone 400 10              ◀ Sound a beep
tlt 10                   ◀ Tell the turtle to turn
wander
end
```

The `tone` primitive takes two inputs. The first input specifies the **pitch** of the tone and the second input specifies the **duration** of the tone (see the *Reference Guide*).

Now put a doorway in the turtle's box so that the turtle can "escape." How long does it take the turtle to escape? Try it several times, and record the results in your Inventor's Notebook.



Challenges

1. Try changing the size of the doorway in the turtle box. Does the size of the door change the turtle's "escape time"?

2. How can you change the `wander` procedure so that the turtle escapes more quickly? (The turtle must use the doorway!)

3. Here's a tough challenge: Program the turtle so that it sounds several beeps and automatically stops when it gets out of the box. (Hint: Use `timer` to keep track of how long it has been since the turtle saw a black line. If it hasn't seen a line in a long time, it must have escaped.)

Challenges

1. If the door is larger, the turtle should escape more quickly. But this is only true on **average**. In any one attempt, the turtle might get lucky and escape through a small door quickly.

2. Try different inputs for `tlt`. Perhaps `tlt 10` works better than `tlt 20`. It is a good idea for students to try several different inputs, and record average escape times for each.

3. Here's one possibility:

```
to wander2
gofd                                      ◀ Go forward
listento 6
resett                                    ◀ Reset the timer
waituntil [or timer > 50 sensor?]         ◀ Wait for five seconds
                                            or until the turtle
                                            sees the black line
if timer > 50 [escaped stop]              ◀ If more than five
                                            seconds, assume the
                                            turtle has escaped
tlt 10                                    ◀ If not escaped, turn
                                            left and . . .
wander2                                   ◀ Wander some more
end

to escaped
offturtle                                 ◀ Turn off the turtle
repeat 3 [tone 400 10]                    ◀ Sound beeps
tone 525 30                               ◀ Final beep
end
```

The turtle with a single touch sensor is not very reliable. Consider what happens when the turtle bumps into a wall at an angle. If the body of the turtle hits the wall before the touch sensor does, the turtle will get "stuck."

With two touch sensors, the turtle is less likely to get "stuck" against a wall. Also, the turtle gets more **information**, since it knows whether it hit the wall with its left bumper or its right bumper.

Sensors. You might discuss other types of sensors that would be useful for the LEGO® turtle. Some possibilities: a sound sensor (like a microphone), a temperature sensor (like a thermometer), a sonar sensor (like an auto-focusing camera).

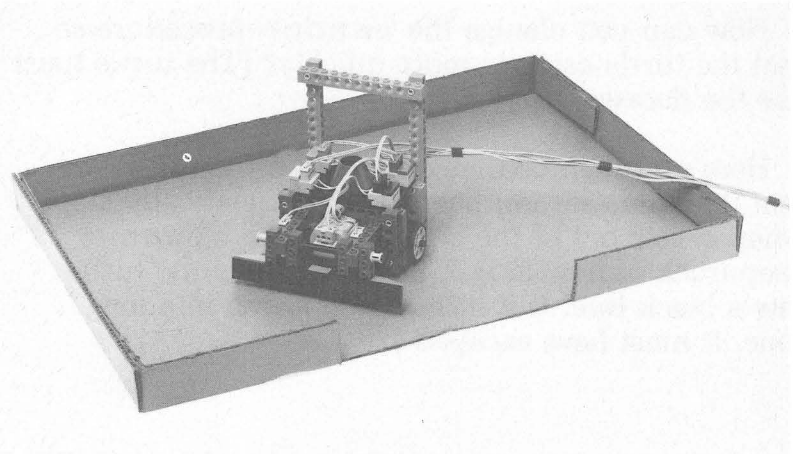
Turtle in a REAL Box

What if you put the LEGO® turtle inside a **real** box? You might use a cardboard box or a wooden crate. How can the turtle know when it bumps into a wall?

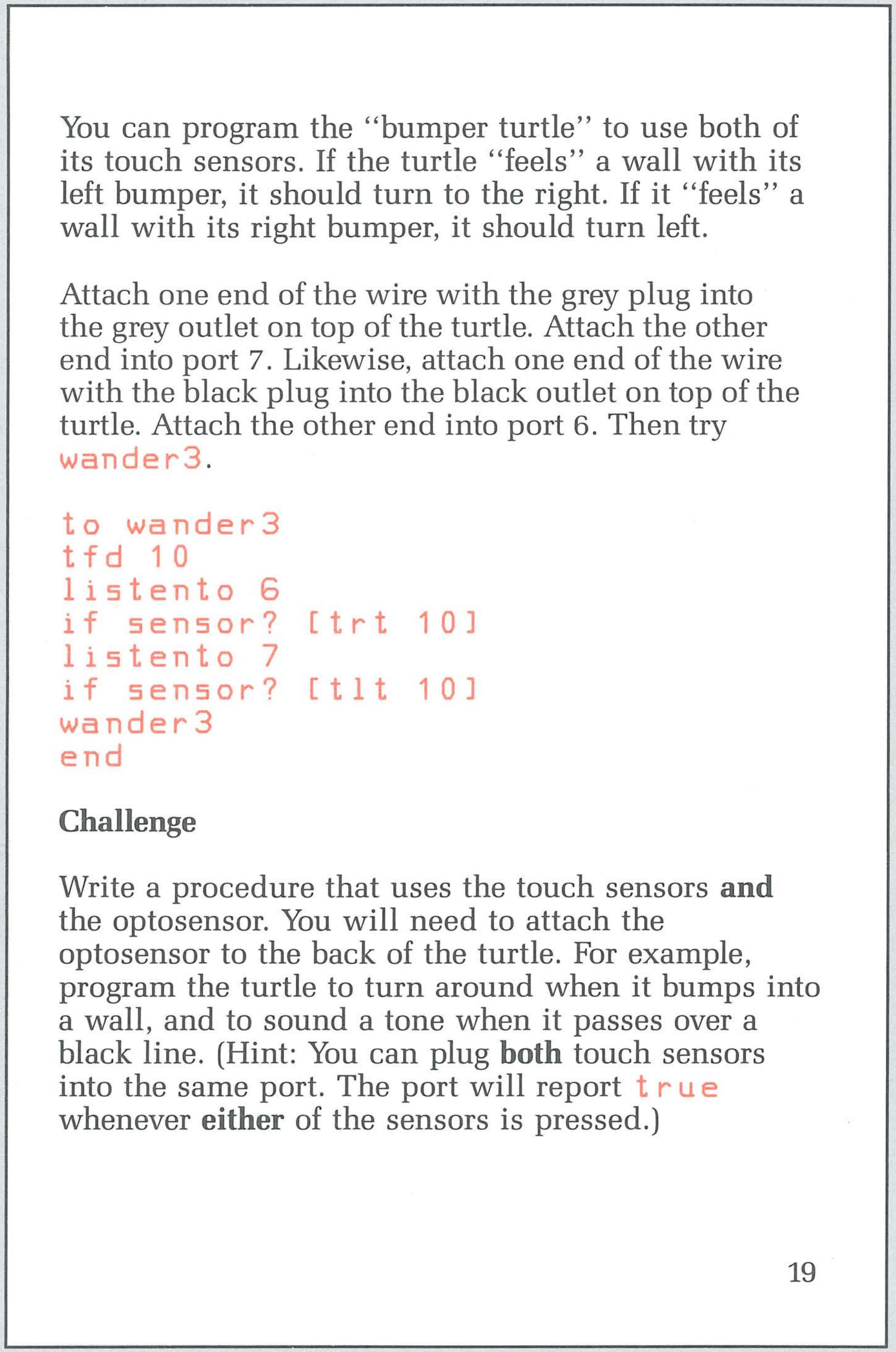
The optosensor will not be much help. The turtle needs **touch sensors**.

If you put a single touch sensor on the front of the turtle, you can use the same `wander` procedure as before (page 16). Try it! When the turtle bumps into a wall, it should turn and start "wandering" some more.

You can make the turtle "smarter" by giving it **two** touch sensors. Activity Card 8 shows how to make two "bumpers" for the turtle, with a touch sensor under each one.



18



You can program the "bumper turtle" to use both of its touch sensors. If the turtle "feels" a wall with its left bumper, it should turn to the right. If it "feels" a wall with its right bumper, it should turn left.

Attach one end of the wire with the grey plug into the grey outlet on top of the turtle. Attach the other end into port 7. Likewise, attach one end of the wire with the black plug into the black outlet on top of the turtle. Attach the other end into port 6. Then try `wander3`.

```
to wander3
tfd 10
listento 6
if sensor? [trt 10]
listento 7
if sensor? [tlt 10]
wander3
end
```

Challenge

Write a procedure that uses the touch sensors **and** the optosensor. You will need to attach the optosensor to the back of the turtle. For example, program the turtle to turn around when it bumps into a wall, and to sound a tone when it passes over a black line. (Hint: You can plug **both** touch sensors into the same port. The port will report `true` whenever **either** of the sensors is pressed.)

19

Mazes. You can set up a maze, then program the LEGO® turtle to find its way through the maze. You could make the maze with black tape on a white surface (and use the optosensor on the turtle). Or you could make the maze out of wood or cardboard boxes (and use the touch sensor on the turtle).

Challenge

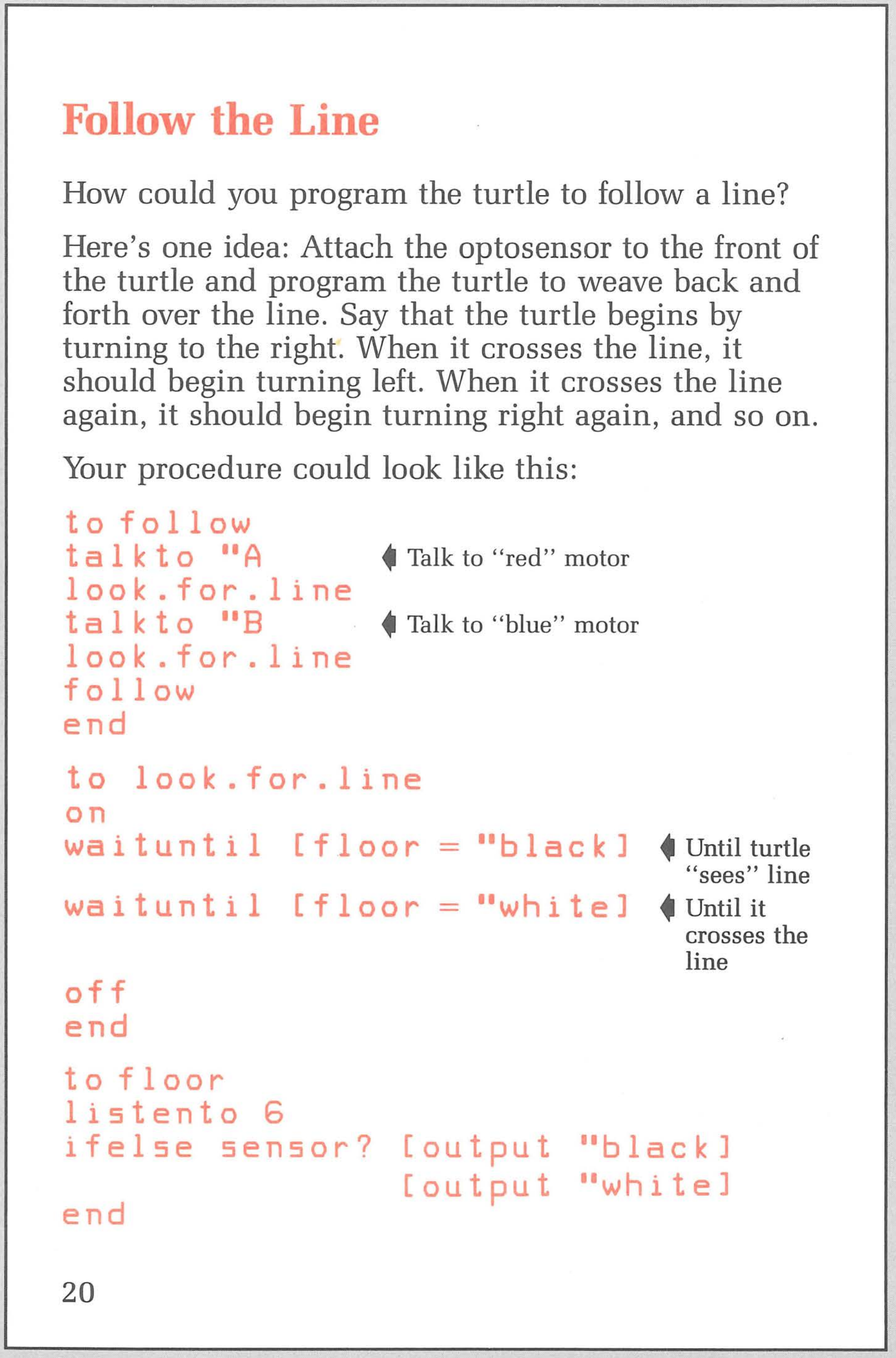
Here's one possibility. The procedure assumes that both touch sensors are plugged into port 6 and the optosensor is plugged into port 7.

Attach one end of a long wire with a black plug into port 6. Connect the other end into the black outlet on the turtle. Attach one end of a long wire with a gray plug into port 7. Connect the other end into the gray outlet on the turtle. Take a short wire and attach one end into the gray outlet and the other end into the optosensor. Take two short wires and attach them together like an extension cord. Attach the two ends into the two touch sensors. Finally, plug the "doubled-up" middle of the extension cord into the black outlet on the turtle.

```
to two.senses
gofd
listento 6
if sensor? [turn.around]
listento 7
if sensor? [tone 400 10]
two.senses
end

to turn.around
tbk 2

tlt ??
end
```

◀ Turtle may need to back up before it can turn

◀ Fill in the appropriate number



Follow the Line

How could you program the turtle to follow a line?

Here's one idea: Attach the optosensor to the front of the turtle and program the turtle to weave back and forth over the line. Say that the turtle begins by turning to the right. When it crosses the line, it should begin turning left. When it crosses the line again, it should begin turning right again, and so on.

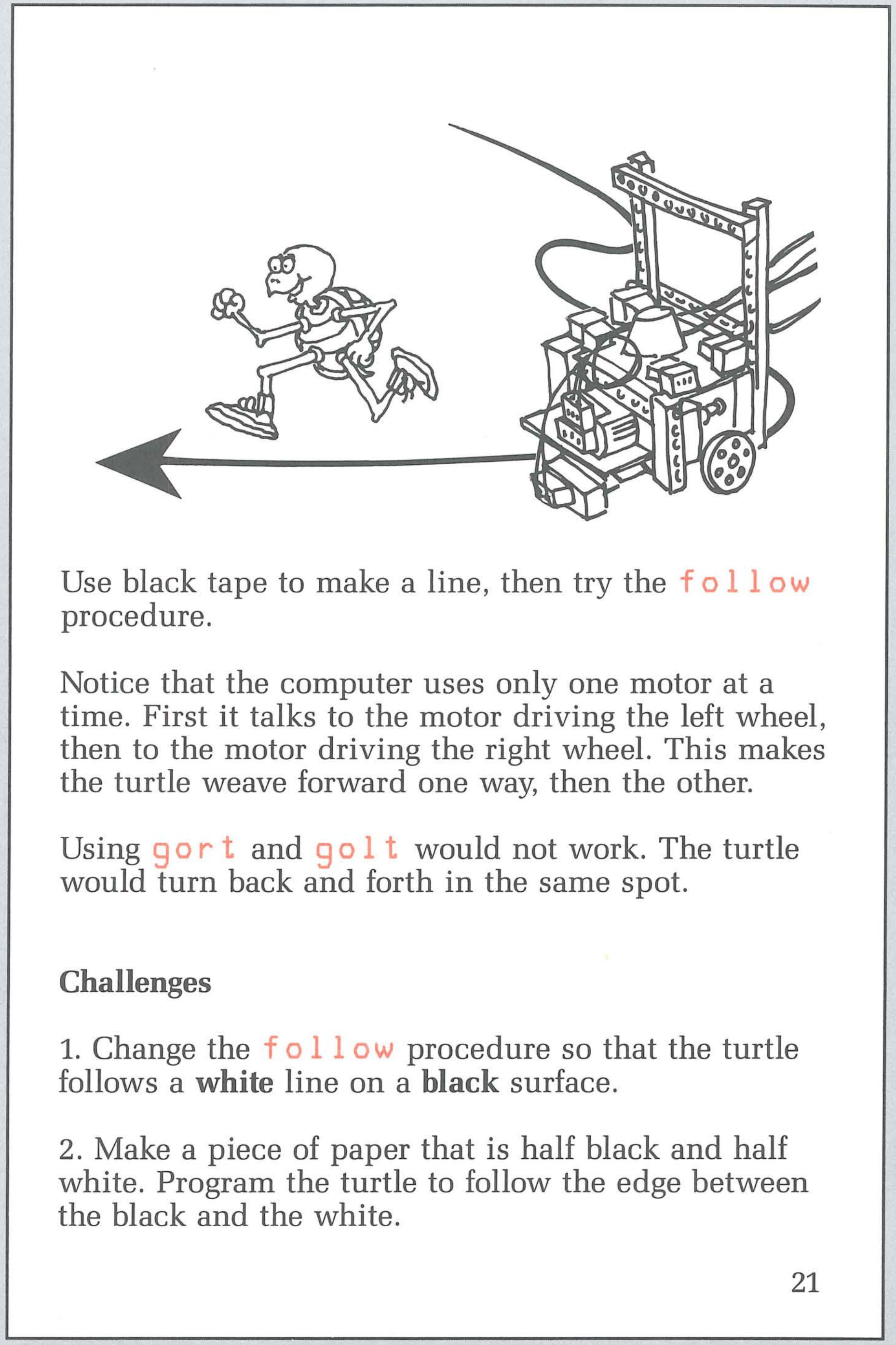
Your procedure could look like this:

```
to follow
talkto "A              ◀ Talk to "red" motor
look.for.line
talkto "B              ◀ Talk to "blue" motor
look.for.line
follow
end

to look.for.line
on
waituntil [floor = "black]   ◀ Until turtle
                               "sees" line
waituntil [floor = "white]   ◀ Until it
                               crosses the
                               line
off
end

to floor
listento 6
ifelse sensor? [output "black]
               [output "white]
end
```

20

You do not really need the `floor` procedure. But it makes the `look.for.line` procedure more "readable." Without `floor`, you would have:

```
to look.for.line
on
listento 6
waituntil [sensor?]
waituntil [not sensor?]
off
end
```



Use black tape to make a line, then try the `follow` procedure.

Notice that the computer uses only one motor at a time. First it talks to the motor driving the left wheel, then to the motor driving the right wheel. This makes the turtle weave forward one way, then the other.

Using `gort` and `golt` would not work. The turtle would turn back and forth in the same spot.

Challenges

1. Change the `follow` procedure so that the turtle follows a **white** line on a **black** surface.

2. Make a piece of paper that is half black and half white. Program the turtle to follow the edge between the black and the white.

When you turn on just one motor, the turtle pivots about the non-moving wheel. `gort` works differently. It turns on both motors—but in opposite directions. This causes the turtle to turn in place, pivoting about its own center. Thus, the turtle does not make any forward progress. (`golt` is similar to `gort`, except each motor turns in the other direction). When you turn on just one motor at a time, the turtle "sweeps" back and forth, making a little forward progress each time.

Challenges

1. Simply change `look.for.line`:

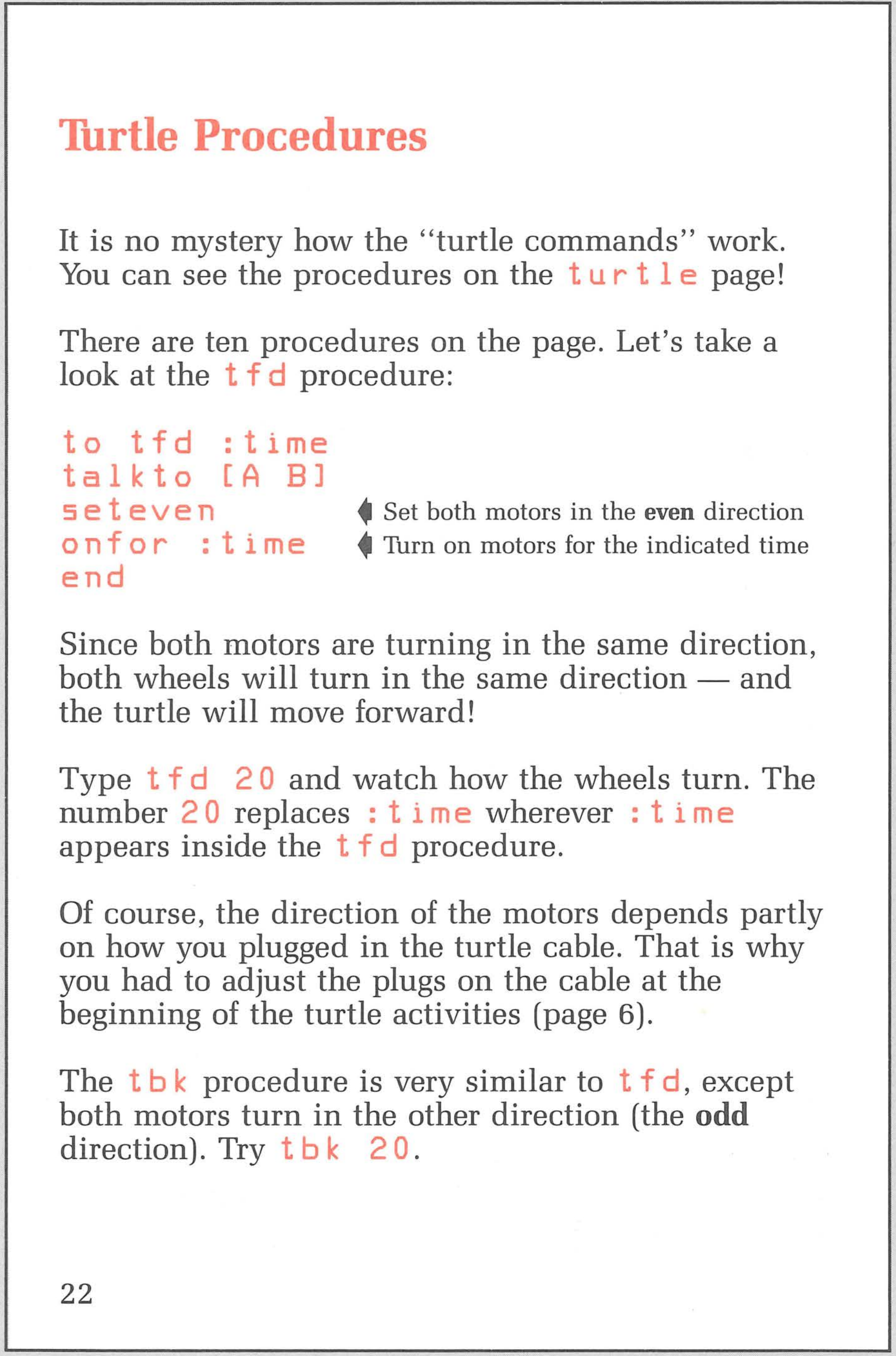
```
to look.for.line
on
waituntil [floor = "white]
waituntil [floor = "black]
off
end
```

The procedure works best if you use bright white tape on a dull black surface.

2. Here's the plan: make the turtle weave one way until it "sees" black, then the other way until it "sees" white. The procedure could look like this:

```
to straddle
talkto "A
on
listento 6
waituntil [floor = "black]
off
talkto "B
on
waituntil [floor = "white]
off
straddle
end
```

Every motor can turn in two directions. In LEGO® TC logo, one direction is called the **even** direction, the other is called the **odd** direction. For a more complete discussion of `seteven` and `setodd`, see the *Reference Guide.*



Turtle Procedures

It is no mystery how the "turtle commands" work. You can see the procedures on the `turtle` page!

There are ten procedures on the page. Let's take a look at the `tfd` procedure:

```
to tfd :time
talkto [A B]
seteven          ◀ Set both motors in the even direction
onfor :time      ◀ Turn on motors for the indicated time
end
```

Since both motors are turning in the same direction, both wheels will turn in the same direction — and the turtle will move forward!

Type `tfd 20` and watch how the wheels turn. The number `20` replaces `:time` wherever `:time` appears inside the `tfd` procedure.

Of course, the direction of the motors depends partly on how you plugged in the turtle cable. That is why you had to adjust the plugs on the cable at the beginning of the turtle activities (page 6).

The `tbk` procedure is very similar to `tfd`, except both motors turn in the other direction (the **odd** direction). Try `tbk 20`.

22

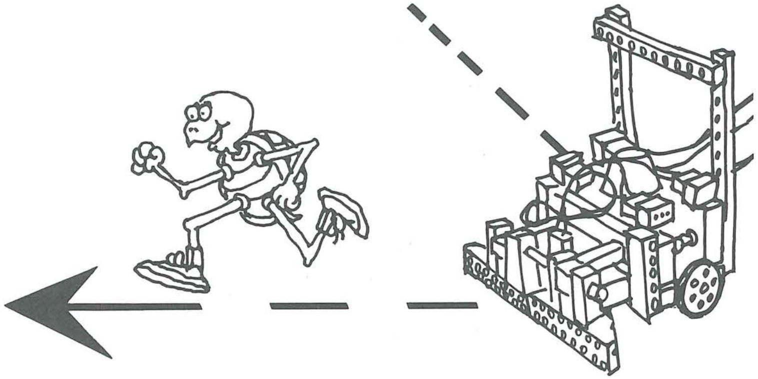
`trt` is a bit trickier:

```
to trt :time
talkto "A       ◀ Talk to the right motor (in port A)
setodd          ◀ Set it in the backwards (odd) direction
talkto "B       ◀ Talk to the left motor (in port B)
seteven         ◀ Set it in the forward (even) direction
talkto [A B]    ◀ Talk to both motors
onfor :time     ◀ Turn them on for :time
end
```

The procedure sets each motor in a **different** direction. So one wheel will turn forward and the other wheel will turn backward. That makes the turtle turn in place.

`tlt` is similar to `trt` except that each motor (and each wheel) turns in the opposite direction.

Try `trt 20` and `tlt 20` and watch the wheels turn.



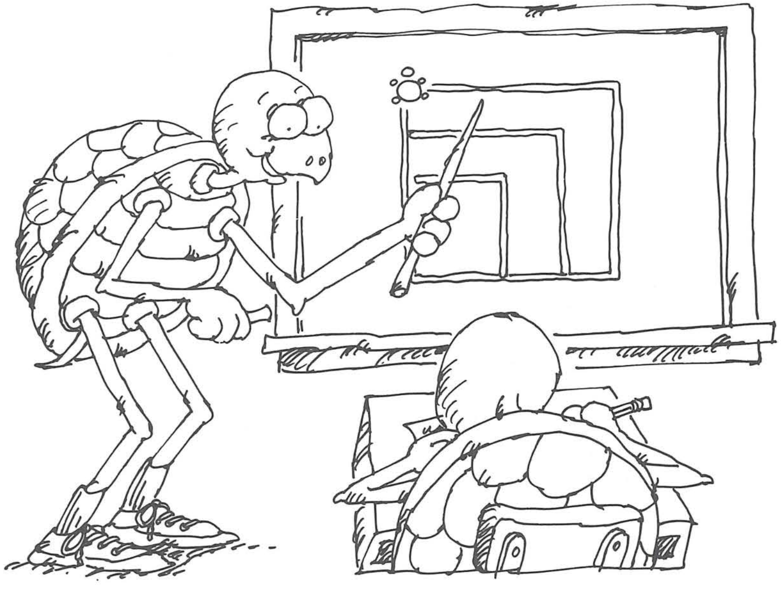
You could just use `alloff` instead of `offturtle`. But for activities involving the LEGO® turtle, the name `offturtle` is more meaningful.

The next four procedures (`gofd`, `gobk`, `gort`, `golt`) are very similar to the first four. There is only one change: They all use `on` rather than `onfor`.

These procedures do not tell the turtle how long to move. They simply start the turtle moving. The turtle will continue moving in the same way until it is told to do something else.

The last two procedures (`offturtle` and `ot`) tell the turtle to stop. They simply turn off the turtle's two motors.

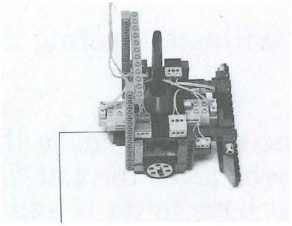
Try the procedures `gofd`, `gobk`, `gort`, `golt`, and `offturtle`.

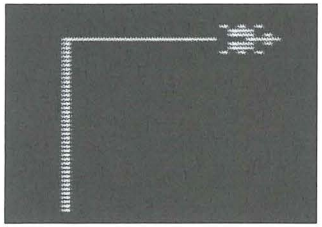


Simulating the Turtle

Imagine that you had a very long turtle cable and you could send the LEGO® turtle into the next room. You would no longer be able to see the turtle. How could you know what the turtle was doing?

One way is to use the "screen turtle" (on the computer screen) to **simulate** the LEGO turtle. Whenever the **LEGO®** turtle goes forward, the **screen** turtle should go forward. Whenever the **LEGO®** turtle turns right, the **screen** turtle should turn right. And so on.





The computer screen is somewhat like a page of paper. The procedures are on one side. The screen turtle "lives" on the other side. To **flip** to the turtle side, hold down the or Ctrl key and press the **F** key (**F** for **F**lip).

You can use the commands `fd`, `bk`, `lt`, and `rt` to move the screen turtle. For example, try these commands:

```
fd 50
rt 90
repeat 4 [fd 50 rt 90]
```

Type `cg` to clear the graphics.

25

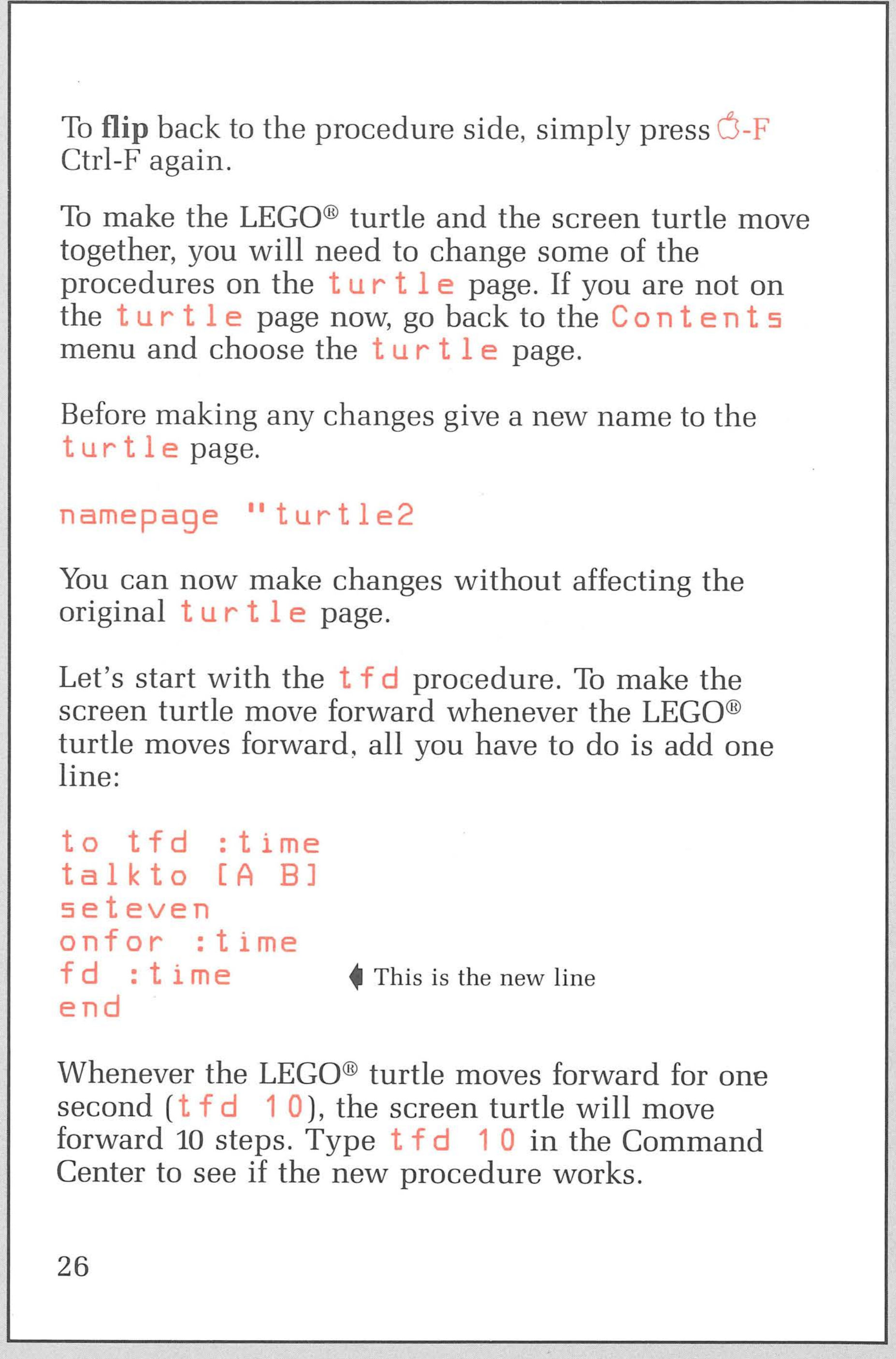
Simulation. There are two types of simulation involved in this activity. First, the LEGO® turtle **simulates** a real turtle. That is, it **mimics** the behavior of a real turtle. Second, the screen turtle simulates the behavior of the LEGO® turtle.

Simulation is a very useful technique. Some simulations let you try things that would be very dangerous in the real world. For instance, the computer can simulate the cockpit of an airplane and let you be the pilot. Or the computer can simulate the meltdown of a nuclear power plant.

Other simulations let you explore processes that would take much longer in the real world. For example, the computer could simulate how the population of the world is going to change in the next 200 years. You can also ask "What if?" kinds of questions: What if the world economy grows strongly? How will that affect the population?

If students have not used Logo graphics before, you might want to give them some time to experiment with the screen turtle.

Similarly, if the LEGO® turtle moves forward for two seconds (`tfd 20`), the screen turtle will move forward 20 steps.



To **flip** back to the procedure side, simply press -F
Ctrl-F again.

To make the LEGO® turtle and the screen turtle move together, you will need to change some of the procedures on the `turtle` page. If you are not on the `turtle` page now, go back to the `Contents` menu and choose the `turtle` page.

Before making any changes give a new name to the `turtle` page.

```
namepage "turtle2
```

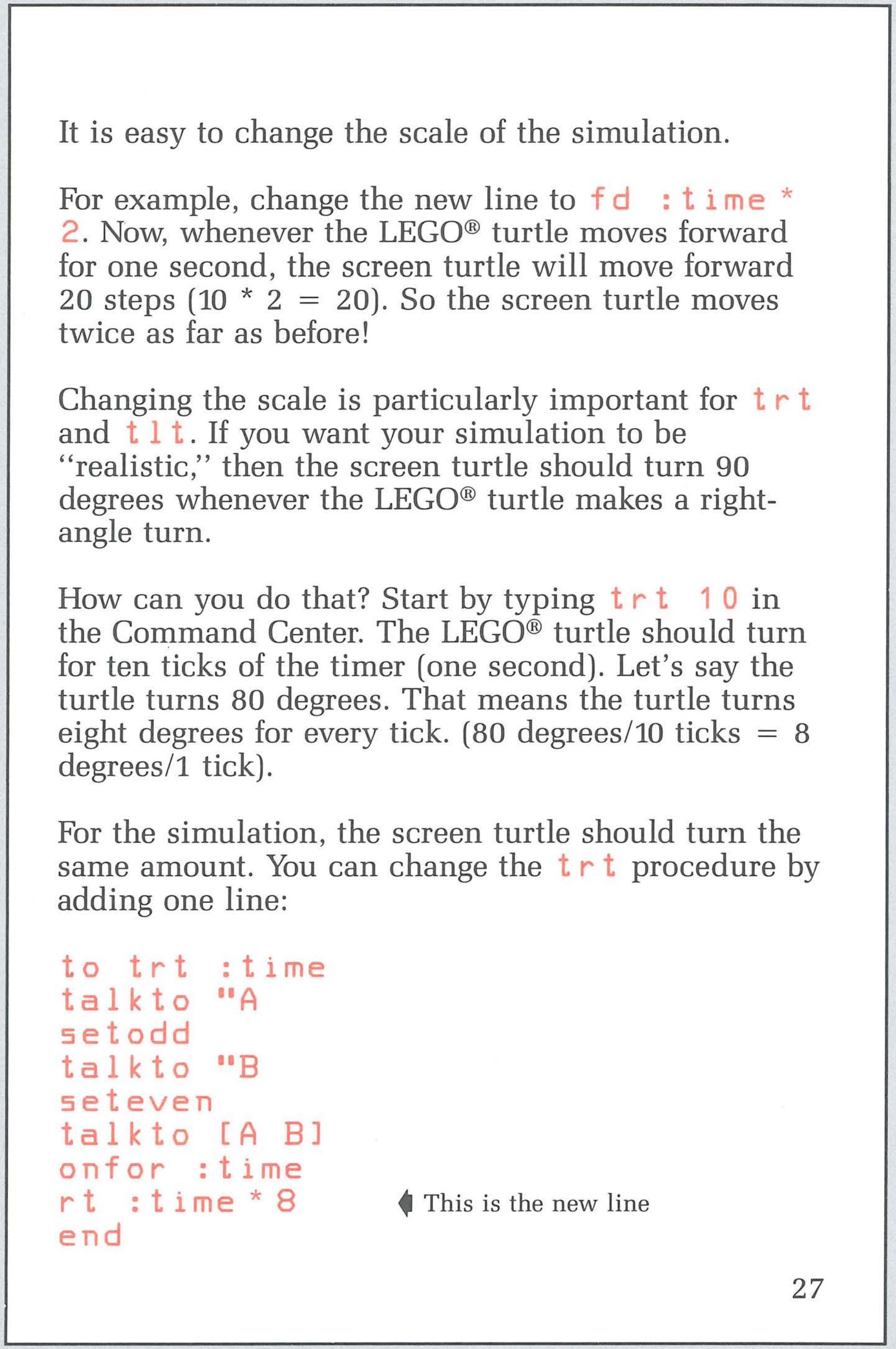
You can now make changes without affecting the original `turtle` page.

Let's start with the `tfd` procedure. To make the screen turtle move forward whenever the LEGO® turtle moves forward, all you have to do is add one line:

```
to tfd :time
talkto [A B]
seteven
onfor :time
fd :time
end
```

◀ This is the new line

Whenever the LEGO® turtle moves forward for one second (`tfd 10`), the screen turtle will move forward 10 steps. Type `tfd 10` in the Command Center to see if the new procedure works.



It is easy to change the scale of the simulation.

For example, change the new line to `fd :time * 2`. Now, whenever the LEGO® turtle moves forward for one second, the screen turtle will move forward 20 steps (10 * 2 = 20). So the screen turtle moves twice as far as before!

Changing the scale is particularly important for `trt` and `tlt`. If you want your simulation to be "realistic," then the screen turtle should turn 90 degrees whenever the LEGO® turtle makes a right-angle turn.

How can you do that? Start by typing `trt 10` in the Command Center. The LEGO® turtle should turn for ten ticks of the timer (one second). Let's say the turtle turns 80 degrees. That means the turtle turns eight degrees for every tick. (80 degrees/10 ticks = 8 degrees/1 tick).

For the simulation, the screen turtle should turn the same amount. You can change the `trt` procedure by adding one line:

```
to trt :time
talkto "A
setodd
talkto "B
seteven
talkto [A B]
onfor :time
rt :time * 8          ◀ This is the new line
end
```

If you change the line to

`fd :time * 0.5` or `fd :time / 2`

the screen turtle will move forward only **five steps** whenever the LEGO® turtle moves forward for one second.

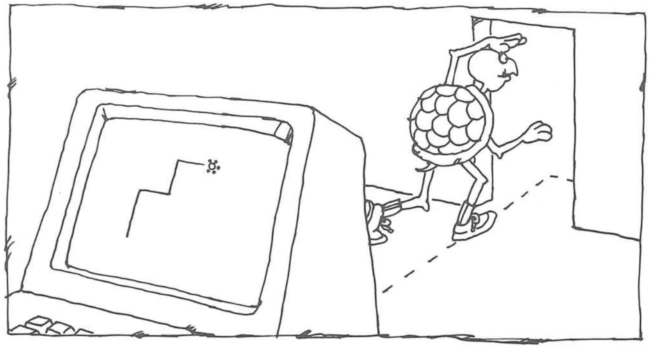
Your new line will probably be different. For example, if `trt 10` makes the LEGO turtle turn 70 degrees, then you should use `rt :time * 7`. Make sure to leave spaces before and after the `*` symbol.

Challenges

1. `tbk` needs a new line similar to the new line in `tfd` (but using `bk` instead of `fd`). `tlt` needs a new line similar to the new line in `trt` (but using `lt` instead of `rt`).

2. The `square` procedure should be the same as before. See page 117 of the *Teacher's Guide.*

3. See page 9 of *Teaching the Turtle.*



Now, if you type `trt 10`, the input to `rt` will be 10 * 8 = 80. So the screen turtle will turn 80 degrees, just like the LEGO® turtle!

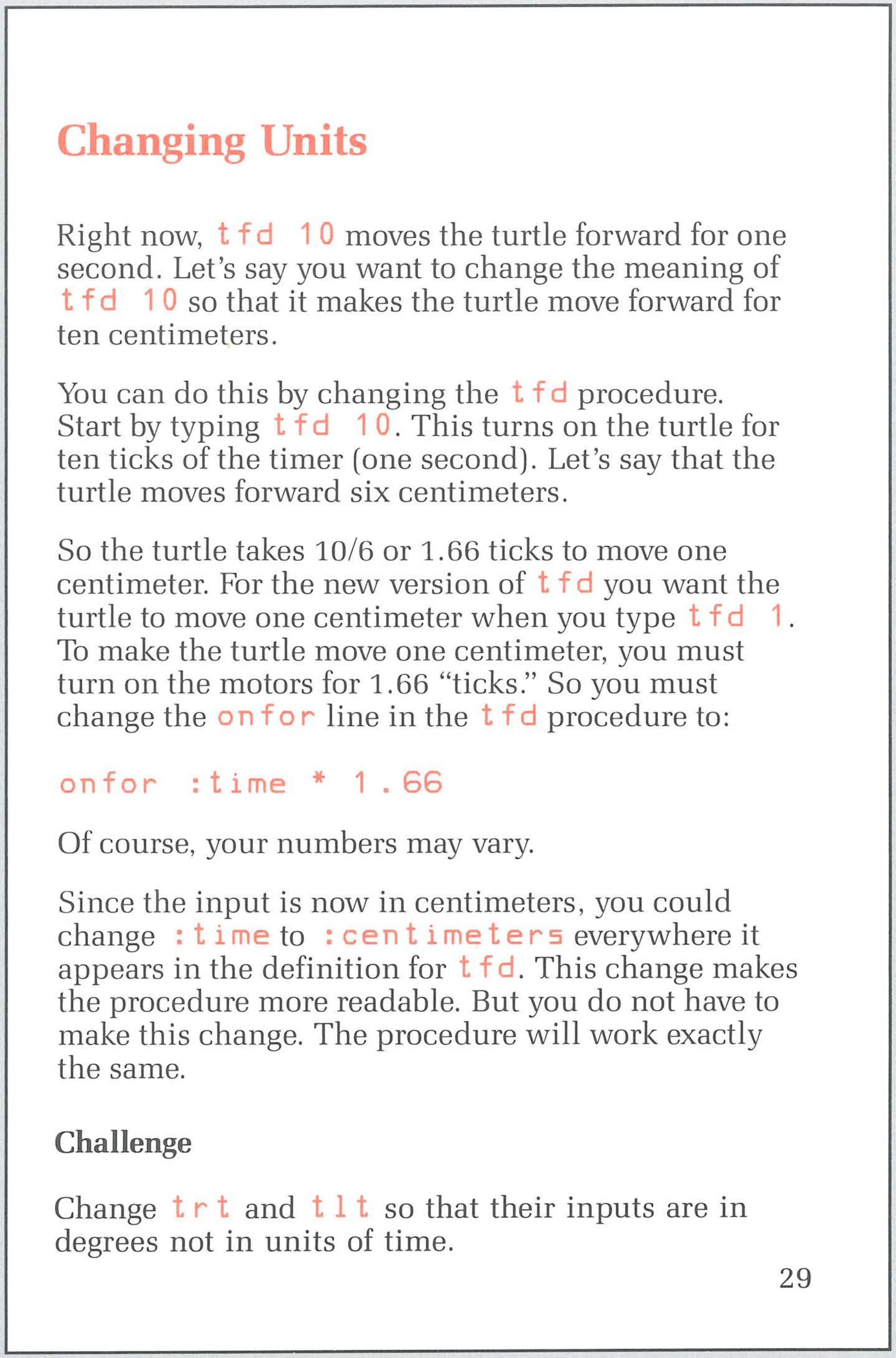
Unfortunately, it is very difficult to make the simulation work for `gofd`, `gobk`, `golt`, and `gort`. These commands start the LEGO® turtle moving and leave it moving. It is difficult to simulate that with the screen turtle.

Challenges

1. Use the same approach to change `tbk` and `tlt`. When you are finished, the screen turtle should act just like the LEGO® turtle!

2. Write a `square` procedure so that the LEGO® turtle and the screen turtle both draw squares.

3. Put the LEGO® turtle back into your turtle town (page 8). Try the `home.to.school` procedure again. You can make a "blueprint" of your turtle town on the screen.



Changing Units

Right now, `tfd 10` moves the turtle forward for one second. Let's say you want to change the meaning of `tfd 10` so that it makes the turtle move forward for ten centimeters.

You can do this by changing the `tfd` procedure. Start by typing `tfd 10`. This turns on the turtle for ten ticks of the timer (one second). Let's say that the turtle moves forward six centimeters.

So the turtle takes 10/6 or 1.66 ticks to move one centimeter. For the new version of `tfd` you want the turtle to move one centimeter when you type `tfd 1`. To make the turtle move one centimeter, you must turn on the motors for 1.66 "ticks." So you must change the `onfor` line in the `tfd` procedure to:

```
onfor :time * 1.66
```

Of course, your numbers may vary.

Since the input is now in centimeters, you could change `:time` to `:centimeters` everywhere it appears in the definition for `tfd`. This change makes the procedure more readable. But you do not have to make this change. The procedure will work exactly the same.

Challenge

Change `trt` and `tlt` so that their inputs are in degrees not in units of time.

29

Challenge

Start by typing `trt 10`. This turns on the turtle for ten time units (one second). Say that the turtle turns 80 degrees. So the turtle turns eight degrees for each time unit.

`Onfor` truncates its input to an integer value so you will lose some precision with the LEGO® turtle.

Now think about the new `trt`. When you type `trt 100`, you want the turtle to turn 100 degrees. Since the turtle turns 8 degrees every time unit, you must turn the motors on for `100 / 8` time units. So you should change the `onfor` line to:

```
onfor :time / 8
```

If you also want to control the screen turtle you should add the line `rt :time` (So the screen turtle will turn 100 degrees when you type `trt 100`.)

The change to `tlt` is similar.

Appendix 1
Worksheets

Soap Box Derby

Sketches of Cars:

Car Number	Description of Car	Distance	Average Distance	Car Number	Description of Car	Distance	Average Distance
1				5			
2				6			
3				7			
4				8			

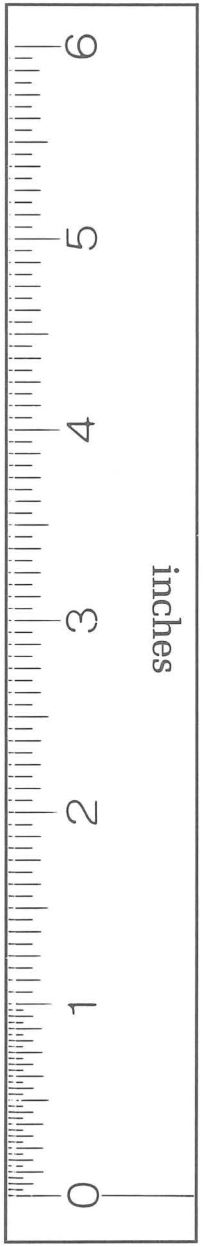
LEGO® TC logo Patent

Sketches/Diagrams:

Description:

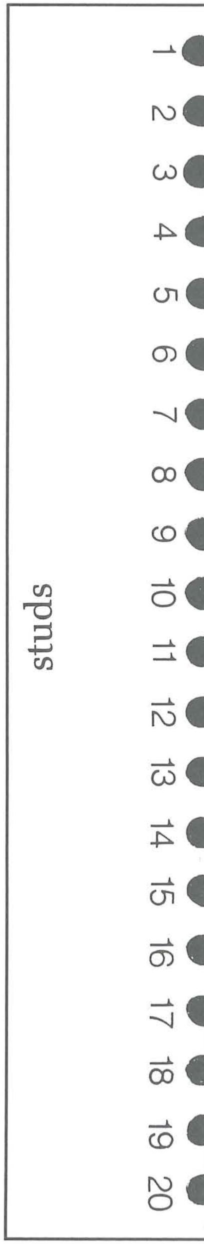
Measurement Worksheet

1 inch = 2.54 centimeters
1 inch = 3.20 1 x 1 stud LEGO® bricks
1 centimeter = 0.39 inches
1 centimeter = 1.27 LEGO studs
1 x 1 stud LEGO brick = 5/16 inch
1 x 1 stud LEGO brick = .79 centimeters

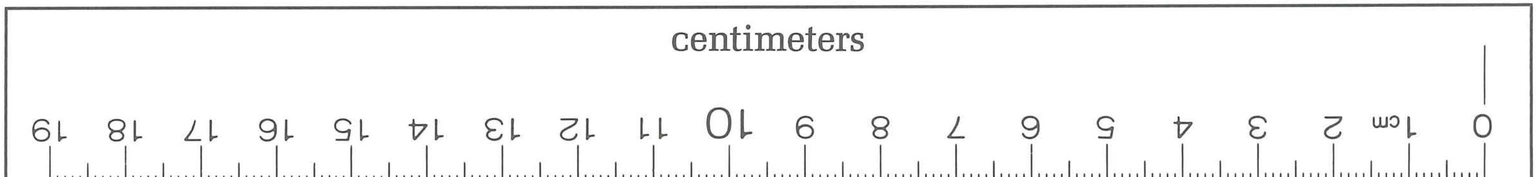


English Ruler

LEGO Ruler



Metric Ruler



Bibliography

I. Computers

Hawkes, Nigel ***Computers and How They Work*** New York: Franklin Watts. 1983.	Good introduction to the parts of a computer system.
Hawkes, Nigel ***Computers in Action*** New York: Franklin Watts. 1983.	How computers are used in the world.
Litterick, Ian ***The Age of Computers: Robots and Intelligent Machines*** New York: Bookwright Press. 1984.	Excellent illustrations.

II. Robots

Milton, Joyce ***Here Come the Robots*** New York: Hastings House Publ. 1981.	
Knight, David C. ***Robotics: Past, Present, and Future*** New York: William Morrow & Co. 1983.	Best for teachers and older students (grades 6-8).
Paltrowitz, Stuart and Donna ***Robotics*** New York: Julian Messner. 1983.	

III. How machines work

Cooper, Chris *How Everyday Things Work* London: Orbis. 1984.	Good section on gears.
Hellman, Hal *The Lever and the Pulley* New York: M. Evans & Co. 1971.	Excellent illustrations.
Keen, Martin L. *How It Works* New York: Grosset and Dunlap. 1974.	Advanced reading level, but excellent resource.
Macaulay, David *The Way Things Work* Boston: Houghton Mifflin. 1988.	
Niel, Ardley *Making Things Move* London: Franklin Watts. 1984.	Lots of simple, hands-on experiments.
Sharp, Elizabeth *Simple Machines and How They Work* New York: Random House. 1959.	Lots of simple, hands-on experiments.
Weis, Harvey *How to be an Inventor* New York: Thomas Crowell. 1980.	Excellent illustrations.
Weis, Harvey *Machines and How They Work* New York: Thomas Crowell. 1983.	Excellent illustrations.

IV. Buildings and structures

Kingston, Jeremy ***How Bridges are Made*** New York: Facts on file Publ. 1985.	Great illustrations.
Macauley, David ***Underground*** ***Pyramid*** ***Cathedral*** ***City*** Boston: Houghton Mifflin	Four wonderful essays describing how things got built.
MacGregor, Anne and Scott ***Bridges*** ***Domes*** New York: Lothrop, Lee & Shepard. 1980.	Both books contain lots of hands-on building projects.

V. Cars and other vehicles

Cars

Cole, Joanna ***Cars and How They Go*** New York: Thomas Y. Crowell. 1983.	Good illustrations.
Gaskin, Carol ***A Day in the Life of a Race Car Mechanic*** Mahwah, NJ: Troll Associates. 1985.	

Young, Frank ***Automobile from Prototype to Scrapyard*** New York: Gloucester Press. 1982.	Excellent illustrations and text.

Motorcycles

Barrett, N. S. ***Motorcycles*** London: Franklin Watts. 1984.	Excellent text and photos. Includes glossary.
Jaspersohn, William ***Motorcycle: The Making of a Harley Davidson*** Boston: Little, Brown & Co. 1984.	Photos of all the people involved in making a motorcycle.

Trucks

Barrett, N. S. ***Trucks*** London: Franklin Watts. 1984.	Outstanding book. Includes glossary.
Magee, Doug ***Trucks You Can Count On*** New York: Dodd, Mead & Co. 1985.	Book of photos, with lots of good close-ups.
Tims, Arthur W. ***Finding Out About Trucks*** Hillside, NJ: Enslow Publ. 1981.	Nice, simple drawings.

Big vehicles

Ancona, George ***Monsters on Wheels*** New York: E. P. Dutton & Co. 1974.	Text written for adults, but lots of great photos for students.
Ancona, George ***Monster Movers*** New York: E. P. Dutton & Co. 1983.	

VI. Farm machines

Marston, Hope Irvin ***Machines on the Farm*** New York: Dodd, Mead & Co. 1982.	Great photos
Sheffer, H. R. ***Tractors*** Mankato, MN: Crestwood House. 1983.	A reader. Good text.
Wykeham, Nicholas ***Farm Machines*** Milwaukee: Raintree Children's Books. 1979.	Excellent photos, showing what farm machines do.

VII. Clocks and time

Barland, K. K., and Speicher, H. R. ***Clock, from Shadow to Atom*** Chicago: Follet Publ. Co. 1969.	Great illustrations, but text is a bit advanced.
Johnson, Chester ***What Makes a Clock Tick?*** Boston: Little, Brown & Co. 1969.	Good diagrams.

VIII. Other machines

Han, Christine ***Amusement Park Machines*** Milwaukee: Raintree Children's Press. 1979.	Great photos.
Siegal, Beatrice ***The Sewing Machine*** New York: Walker & Co. 1984.	Historical presentation. Great photos.
Zim, Herbert, and Skelly, J. R. ***Hoists, Cranes, and Derricks*** New York: William Morrow & Co. 1969.	Practical, applied viewpoint. Great illustrations.

IX. Logo

Abelson, Harold
Apple Logo
New York: McGraw-Hill, Inc.

Harvey, Brian
Computer Science, Logo Style
Cambridge, MA: MIT Press. 1985.

Tipps, Steve, et. al.
Nudges: IBM Logo Projects
New York: Holt Rinehart and Winston. 1984.

Logo Exchange: Journal for Logo-Using Educators
Eugene, OR: ICCE, University of Oregon.

X. Resources for teachers

Aylesworth, T. G.
It Works Like This
Garden City, NY: Doubleday & Co. 1967.

Jollands, David
Machines, Power, and Transportation
New York: Arco Publishing. 1984.
Like an encyclopedia. Nice illustrations.

Papert, Seymour
Mindstorms
New York: Basic Books, Inc. 1980.
Educational philosophy of Logo.

Weitzman, David L.
Windmills, Bridges, and Old Machines
New York: Charles Scribners & Sons. 1982.
Great section on bridges.

Walton, Harry
The How and Why of Mechanical Movement
New York: Popular Science Publ. Co. 1968.
Excellent background reading.

LEGO® TC logo

LEGO Dacta
Educational Products
555 Taylor Road
Enfield, CT 06082

LEGO Australia Pty. Ltd.
P.O. Box 639
Lane Cove
N.S.W. 2066

ISBN 0-914831-71-2
198326
Printed in U.S.A.